UNIVERSIDAD AUTÓNOMA DE MADRID

ESCUELA POLITÉCNICA SUPERIOR

Grado en Ingeniería en Tecnologías y Servicios de Telecomunicación

TRABAJO FIN DE GRADO

Diseño de un modelo HIL como periférico en un procesador empotrado

Beatriz Cámara de la Peña
Tutor: Alberto Sánchez González
Ponente: Ángel de Castro Martín

Julio 2017

Diseño de un modelo HIL como periférico en un procesador empotrado

AUTOR: Beatriz Cámara de la Peña
TUTOR: Alberto Sánchez González

Dpto. C-115

Escuela Politécnica Superior

Universidad Autónoma de Madrid

Julio de 2017

Resumen

El control de convertidores conmutados se implementa cada vez de forma más frecuente mediante reguladores digitales por las características ventajosas que presenta, quedando un sistema mixto por la naturaleza analógica (planta) y digital (control).

Es de gran importancia comprobar el correcto funcionamiento de los reguladores para evitar que algún transitorio genere corrientes o tensiones indeseadas. No basta con cerciorarse de que el regulador es estable en simulación, sino que es necesario comprobar que funciona, una vez implementado el regulador en código o hardware.

Hay diferentes alternativas de simulación para sistemas mixtos comercializadas, pero todos presentan un elevado tiempo de simulación. Para solventar el problema, se puede digitalizar la planta mediante un modelo matemático implementado en hardware y éste se integra en un sistema embebido, quedando un sistema único que permite una emulación en tiempo real con un paso de integración muy bajo; esta técnica de simulación se llama HIL (*Hardware In the Loop*).

Hay varias posibilidades de modelado de la planta pero, como se verá, la aritmética en coma fija parametrizable presenta ventajas con respecto a las demás implementaciones, por ello es la que utiliza el modelo que se integra en este Trabajo de Fin de Grado. La principal ventaja de la coma fija parametrizable es que puede configurarse en tiempo de ejecución (sin necesidad de resintetizar el modelo) para poder adaptarse a condiciones diferentes de simulación (tensiones, corrientes, frecuencias de conmutación, etc.). Sin embargo, el principal inconveniente es que el modelo precisamente debe configurarse con la posición de la coma en cada variable, característica que no es necesaria en el caso de la coma flotante.

El objetivo principal de este proyecto es dotar de comunicación al sistema y que el usuario sea capaz de configurar el modelo de la planta a través del ordenador, aprovechando así las características de la coma fija parametrizable.

El trabajo presenta el diseño de un modelo para un convertidor de potencia con diferentes aritméticas, la integración del modelo de planta como un periférico de un microprocesador en una FPGA, y la comunicación para poder configurar la planta. En los resultados se pueden observar distintas emulaciones, configuradas por el usuario por el puerto serie, sin haber resintetizado el modelo.

Abstract

Control of switched-mode converters is increasingly being implemented with digital controllers digitally given the advantageous characteristics it presents, leaving a mixed system by the analogical (plant) and digital (control) nature.

It is of great importance to check the correct operation of the plant regulators to avoid any damage caused by an undesired current trip. It is not enough to make sure that the regulator does not fail individually but to check that it works risklessly with the plant and all the necessary components.

There are different simulation alternatives for mixed systems commercialized, but all of them have a high simulation time. To solve the problem, the plant can be digitalized using a mathematical model implemented in hardware that is integrated into an embedded system; leaving a unique system that allows a real-time emulation with a very low integration step; this simulation technique is called HIL (*Hardware In the Loop*).

There are several possibilities of modeling the plant, but parametric fixed point arithmetic presents advantages over other implementations, so it is the one used by the model that is integrated in this thesis. The main advantage of parametric fixed point arithmetic s that it can be configured at runtime (without the need to resynthesize the model) to be able to adapt to different simulation conditions (voltages, currents, switching frequencies, etc.). However, the main drawback is that the model must precisely be configured with the position of the point in each variable, a characteristic that is not necessary in the case of the floating point.

The main objective of this project is to provide communication to the system and give the user the ability to configure the model of the plant through the computer; taking advantage of the characteristics of the parametric fixed point.

This work presents the design of a power switch with different arithmetic, the integration of a plant model in an embedded system with a microprocessor in an FPGA and the communication to be able to configure the plant. The results show different emulations, configured by the user, without having resynthesized the model.

Palabras clave

Control digital, convertidor de potencia, simulación, *hardware in the loop*, puente en H, emulación.

Keywords

Digital control, power converter, simulation, hardware in the loop, full-bridge, emulation.

Agradecimientos

Para empezar, quería agradecer a mi tutor, Alberto Sánchez, el apoyo, las correcciones y sobre todo las ayudas constantes durante todo el proyecto. También dar las gracias a Ángel de Castro por conducirme hasta este trabajo.

Dar las gracias a mi familia por confiar en mi día tras día durante estos cuatro años. En especial a mis padres y a mi hermano, que siempre me han ayudado a sacar fuerzas en las situaciones más complicadas y me han aguantado en mis peores días.

Por último, dar las gracias a mis compañeros y amigos que han hecho que este camino no haya sido tan duro como parecía.

ÍNDICE DE CONTENIDOS

ÍNDICE DE FIGURAS

ÍNDICE DE TABLAS

1 Introducción

1.1 Motivación

Los convertidores de potencia son circuitos eléctricos analógicos que se encargan de transformar energía de una entrada a una salida, cambiando la tensión, corriente, o incluso de continua a alterna o viceversa. Sus aplicaciones son muchas gracias a la gran variedad de convertidores que existen y los dispositivos que necesitan unas condiciones de alimentación muy específicas.

Dentro de los convertidores de potencia se encuentran las fuentes conmutadas. Éstas necesitan un controlador, cada vez más frecuentemente digital, porque presenta características ventajosas frente al analógico [1], como puede ser la facilidad de uso. El sistema con los controladores digitales es mixto debido a su naturaleza analógica-digital; para diseñar este tipo de sistema es esencial su simulación para asegurarse de que el sistema no presenta errores. Esta comprobación es tan importante porque si se da cualquier error en su diseño y se conecta con la planta analógica real se pueden producir daños materiales que implican los económicos, así como daños personales debidos a disparos de la corriente superando la máxima permitida por la planta.

La depuración del control permite realizar cualquier tipo de pruebas como, por ejemplo, cambios en ciclos de trabajo, sin que el sistema real sufra ningún daño. No solo hay que comprobar el funcionamiento del regulador, sino del sistema completo. Aunque hay herramientas para conseguir un modelo matemático, este puede presentar fallos de implementación destruyendo así el convertidor.

Existen diferentes alternativas de simulación para la depuración de un sistema formado por una planta analógica regulada por un control digital. Entre ellas, está la simulación mixta que permite que la planta y el regulador se simulen a la vez, pero el tiempo de simulación es elevado.

Para combatir el problema del elevado tiempo de simulación se está llevando a cabo la simulación con un modelo HIL, que consiste en implementar el modelo en hardware digitalizando la planta analógica mediante un modelo matemático de manera que en la simulación el regulador no sepa si está controlando una planta real o su modelo digitalizado. Con el modelo HIL se consiguen emulaciones en tiempo real con un paso de integración muy bajo, lo que permite que la resolución de las señales sea alta.

1.2 Objetivos

El objetivo principal de este Trabajo de Fin de Grado es dotar de comunicación al modelo digitalizado de una planta regulada por un controlador digital para que el usuario pueda configurarla y realizar emulaciones en tiempo real gracias a la simulación HIL. Para ello se consideran los siguientes requisitos:

- Se diseña un modelo de la planta en el lenguaje de descripción de hardware VHDL (*VHSIC Hardware Description Languaje)*, con dos tipos de aritméticas distintas, coma flotante no sintetizable (tipo de datos *real*) y coma fija (tipo de datos *sfixed*). Estas implementaciones se desarrollan para conocer bien sus limitaciones y entender por qué finalmente no se implementan en hardware para su simulación en tiempo real.

- Integrar el modelo matemático de la planta como periférico en un sistema con un procesador embebido. Las salidas de la planta se tienen que convertir a analógico para poder sustituir el modelo por la planta analógica real tras su simulación, porque es así como cualquier control de una planta espera que estén las señales sensadas. La integración se realizará sobre un modelo realizado en coma fija parametrizable, que es una evolución del modelo en coma fija *sfixed*. Además, este modelo contará con la emulación de las pérdidas eléctricas de primer orden, por lo que su configuración implicará numerosos parámetros.

- Para poder configurar el modelo en coma fija parametrizable es necesario dotar al sistema de comunicación con el usuario, se realiza por el puerto serie y permite que se puedan realizar las emulaciones en tiempo de ejecución.

1.3 Organización de la memoria

La memoria consta de los siguientes capítulos:

- En el segundo capítulo se presenta el estado del arte en el que se comenta la utilidad de los convertidores de potencia, la imprescindible comprobación del regulador junto al modelo de la planta antes de conectarlo al sistema real y las diferentes posibilidades de implementación de la planta en VHDL.
- El tercer capítulo detalla la obtención de las variables de estado de un convertidor conmutado *full-bridge* así como las implementaciones con diferentes aritméticas llevadas a cabo en el proyecto.
- El cuarto presenta la integración del modelo HIL en un sistema con microprocesador; se explica la arquitectura del sistema integrado completo y la comunicación: entre el microprocesador y los dispositivos integrados en la FPGA; entre el microprocesador empotrado y el modelo de la planta y finalmente la del usuario con el sistema para poder configurar el convertidor de potencia.
- En el quinto capítulo se explican los resultados que se han obtenido al conectar las salidas de las señales del sistema a un osciloscopio para ver su comportamiento.
- Finalmente, en el sexto capítulo se explican las conclusiones extraídas y los posibles trabajos futuros.

2 Estado del arte

Cada vez son más los dispositivos electrónicos controlados de forma digital, ya que presentan características ventajosas frente a los analógicos, destacando la facilidad de uso. Por ejemplo, los reguladores de los conmutadores de potencia (dispositivos analógicos) hoy en día casi todos son digitales.

Los convertidores de potencia se encargan de entregar a la salida la corriente y tensión que se requiere. Hay distintos tipos de convertidores destacando: elevador (*boost*), *Cúk*, reductor (*buck*) y puente en H (*full-bridge*), cada uno hace una transformación diferente, de continua a continua o de continua a alterna y viceversa. [2]. La transformación de potencia también la pueden llevar a cabo reguladores lineales, a través de una resistencia variable van disipando energía, pero presentan una eficiencia menor. Los convertidores conmutados, consiguen la salida del sistema esperada mediante interruptores que permiten pasar o no a la energía, además, pueden ser reductores o elevadores. Los convertidores conmutados pueden transformar la potencia de manera natural, es la alimentación de estos la que se encarga de conectar o desconectar los interruptores; o gracias a una señal de control.

Con los convertidores, que logran la salida especificada por el usuario gracias a un controlador, se forma un sistema de control, que puede ser analógico o digital, pero por las características que ambos presentan la mayoría son digitales. Para diseñar estos sistemas es esencial simular y asegurarse de que el sistema funciona antes de la prueba real. El medio de depuración es indispensable ya que se puede estropear el conmutador si se supera la corriente máxima de los elementos que forman el convertidor por utilizar un control inadecuado. Además, esta subida de corriente origina pérdidas económicas por los daños materiales y puede producir daños a las personas. La depuración del control permite realizar cualquier tipo de pruebas, en cuanto a cambios en ciclos de trabajo, cargas… sin que se dañe el sistema real, es decir, añade versatilidad.

Además de comprobar el funcionamiento del regulador por separado, es necesario asegurarse de que funciona sin riesgo frente a la planta que vaya a controlar y todos los componentes necesarios. Con *Sisotool*, un paquete de *Matlab* [3] se puede conseguir un modelo matemático óptimo, pero al implementarlo se pueden dar errores que impliquen construir el convertidor de cero. Para que esto no ocurra es de gran utilidad la depuración HIL como paso previo a conectar el regulador con una planta real, pudiendo comprobar el correcto funcionamiento del regulador.

Hay diferentes alternativas de simulación para la depuración de un control digital que regula una planta analógica. Una de las alternativas es un simulador mixto digital-analógico, que permite que el regulador y la planta se simulen a la vez [4,5,6]; en [7] se utiliza para un convertidor analógico definido con *Spice* y un controlador en HDL (*Hardware Description Languaje)*. El simulador mixto presenta el importante inconveniente del elevado tiempo de simulación. Otra alternativa que se presenta en [7] es una simulación mixta pero utilizando modelos de *Spice* para acelerar las simulaciones, pero siguen siendo demasiado lentas. Una opción más, sería utilizar un simulador para el control digital y otro para la planta analógica, en esta simulación el diseñador debe crear una interfaz de comunicación entre ambos simuladores [8].

Otra propuesta consiste en implementar en hardware el modelo en lugar de en software, como proponían las opciones anteriores. Esta alternativa consiste en digitalizar la planta analógica para poder implementarla en una FPGA, ordenador o microprocesador disminuyendo el tiempo de simulación [9,10,11]. Con este diseño el modelo se simplifica porque su función es verificar que el sistema final funciona correctamente en lugar de medir su estabilidad y respuesta. La simulación puede producirse como sistema único, si todo está en digital, o a través de DACs (*Digital to Analog Converter)* que permitan la emulación del funcionamiento de la planta analógica y el control digital. El uso de FPGAs da lugar a simulaciones con mayor precisión, ya que permiten reducir al máximo el paso de integración [12,13].

Para modelar los convertidores conmutados en una FPGA hay que implementar las ecuaciones características de la planta eligiendo una aritmética adecuada además de un regulador digital ejecutado en un procesador embebido dentro de la FPGA. La elección de la aritmética es un proceso importante, ya que de ella dependen, entre otras cosas, la resolución y la velocidad de simulación.

Si el modelo de la planta se va a desarrollar en leguaje HDL hay diferentes posibilidades de implementación. Por un lado, los convertidores en coma flotante, que si se especifica el lenguaje de descripción hardware a VHDL hay dos posibles aritméticas: coma flotante no sintetizable, que utiliza el tipo de datos *real*, con el gran inconveniente de no ser sintetizable, por lo tanto, no puede integrarse en hardware, pero tiene una fácil implementación además de una alta resolución gracias al estándar utilizado de 64 bits y doble precisión; por ello se utiliza para hacer comprobaciones con otros modelos que sí son sintetizables. Otro tipo de datos de coma flotante es *float* pero este sí que es sintetizable, aunque con una resolución menor ya que es de 32 bits, la ventaja es que la implementación no es complicada porque la posición de la coma no os objeto de diseño, se adapta automáticamente; pero el tiempo de síntesis es elevado y utiliza muchos recursos.

Otra opción es el diseño del modelo en coma fija, también VHDL presenta dos aritméticas diferentes. La primera es mediante el tipo de datos *sfixed*, que requiere un esfuerzo elevado por parte del diseñador, ya que debe asignar el número de bits necesarios para cada variable, tanto para parte entera como para la fraccionaria; además no es universal, lo que quiere decir que si cambian los valores de los parámetros se tiene que rediseñar el modelo. La ventaja es que solventa el problema del tipo *float* en cuanto a la velocidad de síntesis y utilización de recursos. La otra aritmética es coma fija parametrizable con el tipo de datos *std_logic_vector* que presenta las ventajas del tipo *sfixed* pero la posición de la coma es variable, lo que implica que se pueden realizar varias emulaciones mientras la simulación se está ejecutando. Esta aritmética tiene que configurarse para dotar de comunicación al modelo de la planta en coma fija parametrizable, siendo este el objetivo del Trabajo de Fin de Grado.

3 Diseño de un modelo HIL

En este proyecto, se va a desarrollar la implementación de un modelo extrayendo sus ecuaciones en diferencias a partir del método de Euler explícito. Aunque las variables de estado que se extraen son de un conmutador inversor, hay que destacar que sirve cualquier conmutador de potencia como modelo de planta; de hecho, en el sistema final integrado se utilizará otro convertidor como periférico del microprocesador embebido, se hablará de él más adelante aunque su implementación no es objeto de este trabajo.

3.1 Modelado de un full-bridge

Un puente en H, también conocido como puente completo (o por su término en inglés *full-bridge*) es un circuito electrónico que cambia la tensión de corriente continua a corriente alterna y es capaz de invertir la tensión de entrada. Se utilizan bastante como convertidores de potencia, así como en robótica, ya que permiten que un motor eléctrico de corriente continua gire en sentido de avance y también en retroceso.

Los *full-bridge s*e pueden construir mediante componentes discretos o como circuitos integrados y son capaces de funcionar de manera simultánea con energía eólica y solar fotovoltaica y así poder alimentar tanto en continua como en alterna. Este tipo de fuente conmutada es capaz de generar energía eléctrica a partir de energías renovables.

El estado de un conmutador inversor viene marcado por sus interrupciones. Aunque presenta diferentes topologías, se desarrolla una muy simple para dotar de claridad al desarrollo. En la Figura 3.1 se observa que este puente completo está formado por cuatro interruptores, una carga resistiva (se podría utilizar otro tipo de carga sin alterar la funcionalidad del *full-bridge*) y dos elementos pasivos: un condensador y una bobina que proporcionan las variables de estado v_C e i_L respectivamente.

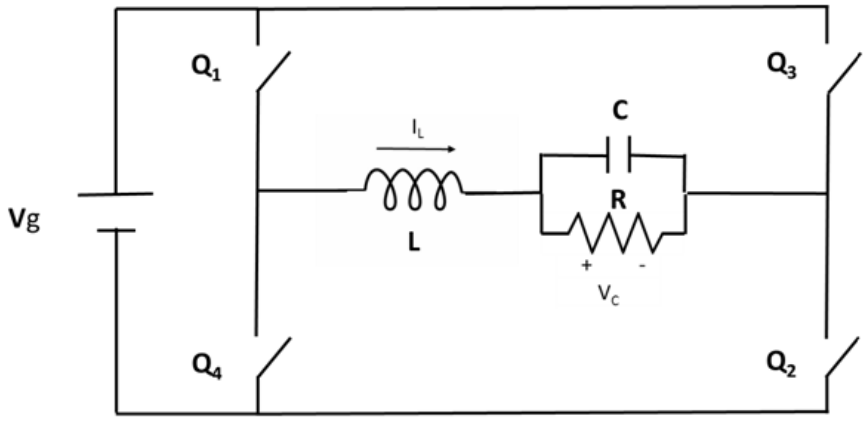

Figura 3-1. Topología *full-bridge* o puente en H

La tensión de salida del circuito corresponde con la que cae en los bornes del condensador, ecuación (3.1)

$$v_{out} = v_c \tag{3.1}$$

Las variables de estado son las que determinan unívocamente el estado del sistema, por ello el modelo debe calcularlas en cada instante. A continuación, se describen las ecuaciones necesarias para llegar a ellas.

La tensión de la bobina se define como:

$$v_L = L \cdot \frac{di_L}{dt} \tag{3.2}$$

Si discretizamos la ecuación (3.2), utilizando el método de Euler explícito, se llega a la ecuación en diferencias:

$$i_L(k) = i_L(k-1) + \frac{\Delta t}{L} \cdot v_L\,(k-1) \tag{3.3}$$

Dónde i_L(k) es la corriente que atraviesa la bobina en el instante k, por ende, k-1 es el instante anterior, Δt es el paso de integración para calcular las variables de estado y L es el valor de la inductancia de la bobina.

De manera equivalente la corriente que pasa por el condensador se define como:

$$i_C = C \cdot \frac{dv_c}{dt} \tag{3.4}$$

Si se transforma (3.4) en una ecuación diferencial, mediante el método mencionado anteriormente, se puede obtener la variable de estado proporcionada por el condensador en un instante k:

$$v_C(k) = v_C(k-1) + \frac{\Delta t}{C} \cdot i_C(k-1) \tag{3.5}$$

Una vez determinadas las ecuaciones que definen el estado del puente en H, se va a distinguir estados distintos del modelo que vienen delimitados por la posición de los interruptores.

Los dos estados principales del *full-bridge* se ilustran en la Figura 3.2 y la Figura 3.3. La única diferencia entre ambos es la tensión en bornes de la bobina: en el caso en que los interruptores Q_1 y Q_2 están cerrados y Q_3 y Q_4 abiertos (Figura 3.2) $v_L = v_G - v_C$, ecuación (3.6); es decir, se aplica una tensión positiva. En el caso contrario, Q_1 y Q_2 abiertos y Q_3 y Q_4 cerrados (Figura 3.3) $v_L = -v_G - v_C$, ecuación (3.7), en este estado la tensión del generador se invierte. En cambio, la corriente que atraviesa el

condensador ($i_C = i_L - i_R$) es independiente del estado de los interruptores, por ello la tensión de salida depende únicamente de esta corriente.

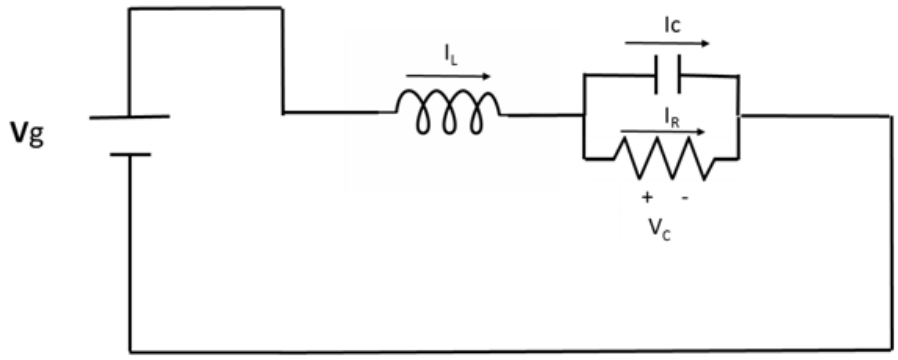

Figura 3-2. Topología *full-bridge*, Q_1 y Q_2 cerrados, Q_3 y Q_4 abiertos

$$i_L(k) = i_L(k-1) + \frac{\Delta t}{L} \cdot (v_G(k-1) - v_C(k-1))$$

$$v_C(k) = v_C(k-1) + \frac{\Delta t}{C} \cdot (i_L(k-1) - i_R(k-1)) \qquad (3.6)$$

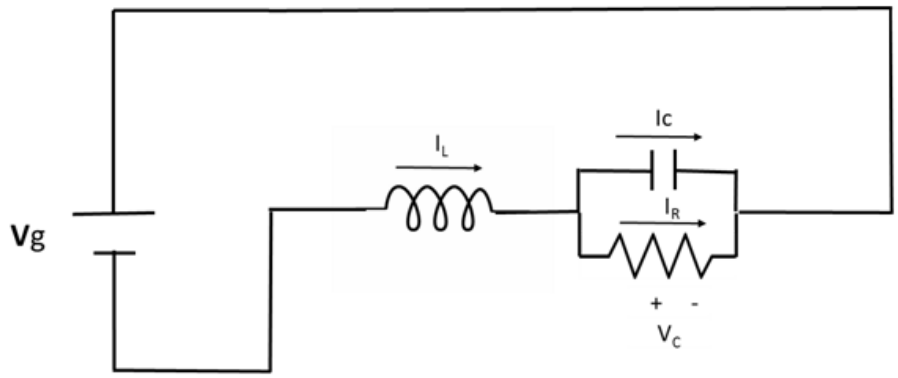

Figura 3-3. Topología *full-bridge*, Q_3 y Q_4 cerrados Q_1 y Q_2 abiertos

$$i_L(k) = i_L(k-1) + \frac{\Delta t}{L} \cdot (-v_G(k-1) - v_C(k-1))$$

$$v_C(k) = v_C(k-1) + \frac{\Delta t}{C} \cdot (i_L(k-1) - i_R(k-1)) \qquad (3.7)$$

Además de estos dos estados, se puede dar el caso en el que todos los interruptores estuvieran cerrados a la vez, lo que produciría un cortocircuito; por otro lado, si se

cerraran con unas combinaciones distintas a las descritas anteriormente, se obtendrían estados más peculiares que no se describen por aportar claridad al desarrollo.

Estos interruptores suelen estar acompañados de diodos conectados en antiparalelo para que la corriente pueda circular en sentido inverso; porque cuando se conmute la tensión, la bobina en cortos instantes de tiempo se opondrá a la variación de corriente.

Para las implementaciones del conmutador de potencia desarrolladas a continuación, se utilizan las ecuaciones previas ((3.6) y (3.7)) con alguna adaptación detallada en el siguiente capítulo.

3.2 Implementación

En el apartado anterior se ha descrito detalladamente el proceso de obtención de las variables de estado que se ilustran en la Tabla 3-1. Estas ecuaciones son necesarias ya que se quieren transcribir a lenguaje HDL para poder implementarlas en la FPGA consiguiendo un modelo de la planta digitalizada. Cabe destacar, que las transcripciones a lenguaje de descripción hardware que se desarrollan en este Trabajo de Fin de Grado no son las que finalmente se implementan como modelo de la planta. Aunque hay varios lenguajes de descripción de hardware, como Verilog o ABEL HDL, el que se utiliza es VHDL.

Tabla 3-1. Ecuaciones de las variables de estado del *full-bridge*

Posición interruptores	**Ecuaciones**	
	I_L	V_C
Q_1=Q_2=1 Q_3=Q_4=0	$i_L(k) = i_L(k-1) + \frac{\Delta t}{L}(v_G(k-1) - v_C(k-1))$	$v_C(k) = v_C(k-1) + \frac{\Delta t}{C}(i_L(k-1) - i_R(k-1))$
Q_3=Q_4=1 Q_1=Q_2=1	$v_C(k) = v_C(k-1) + \frac{\Delta t}{C}(i_L(k-1) - i_R(k-1))$	$v_C(k) = v_C(k-1) + \frac{\Delta t}{C}(i_L(k-1) - i_R(k-1))$

Existen diferentes aritméticas de implementación del modelo:

- Coma flotante no sintetizable: VHDL ofrece la aritmética de tipo *real*; el estándar que utiliza es el IEEE 754 de doble precisión, tiene 53 bits de mantisa que permiten realizar el modelo con mucha resolución. Para comenzar a entender la transcripción al lenguaje de descripción hardware es muy útil esta aritmética por su facilidad. Esta opción presenta un inconveniente trivial, que no es sintetizable, es decir, que este modelo no puede implementarse en hardware real, por lo tanto, no se puede utilizar en un sistema HIL. Pese a este inconveniente, es de gran utilidad para comprobar que las ecuaciones en diferencias extraídas con el método explícito de Euler se han obtenido correctamente.
- Coma flotante: el tipo de datos que utiliza es *float* en 32 bits. A pesar de ser sintetizable, pocos sintetizadores comerciales son compatibles con este tipo. Además, la resolución es insuficiente para frecuencias de conmutación

elevadas. Otros inconvenientes de esta aritmética es que utiliza muchos recursos y la frecuencia de funcionamiento es muy lenta. [14]

- Coma fija: en esta aritmética se utiliza el tipo de datos de la librería que proporciona VHDL-2008, *sfixed*, utiliza el formato QX.Y, la cual es sintetizable y combate el problema que presenta el tipo *float* de utilización de muchos recursos y la baja frecuencia de funcionamiento. Esta posible implementación aumenta el trabajo del diseñador, ya que tiene que hacer un estudio de cada señal para fijar la coma y el formato. Este modelo no es universal, es decir, en tiempo de diseño se tienen que establecer los formatos QX.Y de cada señal que presenta la resolución adecuada; si el valor de estos parámetros cambia y por tanto el formato, se tiene que rediseñar el modelo.
- Coma fija parametrizable: en esta posible implementación, el tipo de datos utilizado es *std_logic_vector*, trabaja con vectores simples que pueden interpretarse en complemento a dos y no requieren saber la posición de la coma. La posición de la coma se calcula en tiempo de ejecución gracias a la escala que lleva definida cada señal, quedando entonces definido el formato en QX.Y. Este modelo presenta todas las ventajas de la coma fija del tipo de datos *sfixed* y solventa sus inconvenientes; el modelo se sintetiza tan solo una vez pudiendo realizar el usuario cambios en los parámetros de entrada para diferentes emulaciones. [15]

Una vez presentadas las diferentes aritméticas, se van a explicar las dos que han sido desarrolladas para mayor entendimiento, así como la aritmética en coma fija parametrizable que, aunque no se desarrolla, es la que utilizará el modelo de la planta en el sistema embebido final.

3.2.1 Modelo en coma flotante no sintetizable del puente en H

La primera aritmética implementada es la de señales de tipo *real*, tipo ofrecido por VHDL, es una aritmética en coma flotante no sintetizable de 64 bits de resolución. Su implementación es la más sencilla porque las señales se ajustan automáticamente, es decir, el diseñador no tiene que prestar atención a la posición de la coma ni al tamaño de las señales ya que es el software el que realiza el ajuste. Este tipo de datos utiliza el estándar IEEE 754 de doble precisión, permitiendo realizar el modelo de la planta con mucho margen aun con frecuencias de conmutación elevadas.

El tipo *real* es soportado por la mayoría de simuladores y presenta una gran precisión en los resultados. Pese a estas buenas características tiene un inconveniente importante; la aritmética de este tipo no es sintetizable, lo que quiere decir que no se puede utilizar en un sistema HIL porque no puede implementarse en hardware real.

Que no sea sintetizable no significa que no sea útil, la alta resolución que presentan los resultados facilita la comprobación del correcto funcionamiento del convertidor conmutado, así como la comparación con las demás aritméticas posibles (en este trabajo se desarrolla únicamente aritmética tipo *real* y coma fija).

Para implementar el *full-bridge* es necesaria la transformación en lenguaje VHDL de las ecuaciones definidas en (3.6) y (3.7), para ello se utiliza el hardware de la Figura

3.4. En estas ecuaciones se hace una pequeña adaptación, para facilitar la síntesis, que consiste en definir Δt como un tiempo fijo para que los términos de las ecuaciones que dependen del paso de integración sean constantes.

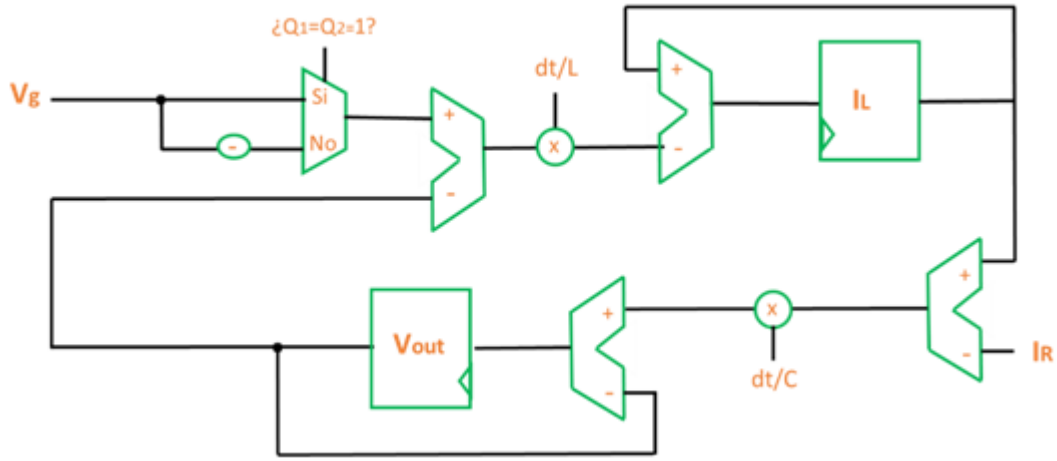

Figura 3-4. Hardware *full-bridge* del modelado en *real*

La declaración de las señales como es de esperar son todas de tipo *real*, se puede observar en la Figura 3.5; sus valores se asignan dentro de la arquitectura. El paso de integración corresponde con “dt” que se ha forzado al tiempo fijo del periodo de reloj del proceso síncrono (Figura 3.6). En este proceso se calculan las variables de estado del modelo.

```
--valores delta
      signal VoDelta    :     real;
      signal IDelta     :     real;

--valores auxiliares salida
      signal Ilaux      :     real := 0.0;
      signal Voaux      :     real := 0.0;

--constantes
      signal C    :     real;
      signal L    :     real;
      signal dt   :     real;
```

Figura 3-5. Declaración de señales, aritmética tipo *real*

```
process (Reset, Clk)
begin
      if (Reset = '1') then
            Voaux <= 0.0;
            Ilaux <= 0.0;
      elsif (rising_edge(Clk)) then
            Voaux <= Voaux + IDelta*dt/C;
            Ilaux <= Ilaux + VoDelta*dt/L;
      end if;
end process;
```

Figura 3-6. Proceso síncrono, aritmética tipo *real*

En el Anexo A se encuentra el código completo de este modelo.

3.2.2 Modelo en coma fija del puente en H

El modelo en coma fija que se describe en esta subsección utiliza la librería *sfixed* de VHDL-2008. Este tipo de aritmética se presenta porque solventa el inconveniente de síntesis del tipo *real*; permite obtener frecuencias de síntesis altas sin utilizar un elevado número de recursos. En la implementación en coma fija se tienen que especificar los tamaños de las señales. El cálculo de esos tamaños requiere un estudio detallado del rango de cada señal para así finalmente fijar el número de bits necesarios tanto para la parte entera como la fraccionaria, consiguiendo la máxima resolución. El inconveniente de la aritmética en coma fija, además del gran esfuerzo del diseñador, es la necesidad de rediseñar el modelo si hay cambios en los rangos de valores reales durante la ejecución del convertidor, ya que podrían desbordar o tener una resolución insuficiente.

La librería *sfixed* utiliza el formato QX.Y, en complemento a 2, el cual utiliza 1 bit de signo, de gran utilidad ya que existe la posibilidad de que algunas tensiones o corrientes sean de sentido opuesto a lo fijado en el modelo; X bits de parte entera e Y bits de parte fraccionaria, como se puede observar en la Figura 3.7.

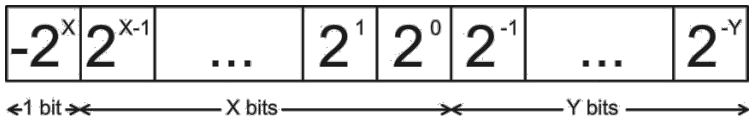

Figura 3-7. Formato de una señal en QX.Y

Un ejemplo de esta notación sería el formato Q4.2, que tendrá un total de 7 bits (1+4+2). Si por ejemplo se tiene el un número en binario: 0101101_2 en Q4.2 sería: 01011.01; si se quiere pasar a decimal hay que seguir la notación en complemento a 2 descrita en la Figura 4.2 ($-0\cdot2^4 + 1\cdot2^3 + 0\cdot2^2 + 1\cdot2 + 1\cdot2^0 + 0\cdot2^{-1} + 1\cdot2^{-2} = 11{,}25_{10}$).

Las operaciones aritméticas necesarias en el modelo son la suma (y resta) y multiplicación, por eso a continuación se detallan los ajustes del formato para estas operaciones. En el caso de una suma, análogamente en una resta, el número de bits de la parte entera corresponde con el mayor tamaño de la parte entera de los sumandos

más uno (para evitar desbordamiento, *overflow* en inglés) y el número de bits de la parte fraccionaria el máximo también de los sumandos de esta parte; para facilitar el entendimiento se adjunta la Figura 3.8. En la multiplicación, el número de bits necesario en la parte entera se calcula sumando los correspondientes a cada multiplicando más uno, por la misma razón que en la suma; el ajuste necesario en la parte fraccionaria es el resultado de sumar los bits de esta parte asignados en los operandos de la multiplicación; la Figura 3.9 facilita la comprensión.

$$\begin{array}{r} Q\ X_1 . Y_1 \\ + \quad Q\ X_2 . Y_2 \\ \hline Q\ \max(X_1, X_2) + 1 . \max(Y_1, Y_2) \end{array} \qquad \begin{array}{r} Q\ 4.2 \\ + \quad Q\ 2.3 \\ \hline Q\ 5.3 \end{array}$$

Figura 3-8. Formato y ejemplo suma QX.Y

$$\begin{array}{r} Q\ X_1 . Y_1 \\ x \quad Q\ X_2 . Y_2 \\ \hline Q\ X_1 + X_2 + 1 . Y_1 + Y_2 \end{array} \qquad \begin{array}{r} Q\ 4.2 \\ x \quad Q\ 2.3 \\ \hline Q\ 7.5 \end{array}$$

Figura 3-9. Formato y ejemplo multiplicación QX.Y

Como en el modelo de tipo *real* las ecuaciones que se transcriben en código VHDL son (3.6) y (3.7). En este modelo se utiliza el mismo hardware que en el modelo en coma flotante no sintetizable, se ilustra en la Figura 3.10 con los formatos, previamente calculados de la manera descrita en el párrafo anterior, que necesita cada señal. El máximo valor admitido por las tensiones es de 64 Voltios, de ahí el formato Q6.10 (2^6 = 64) y el máximo de las corrientes 8 amperios, formato Q3.13 (2^3 = 8).

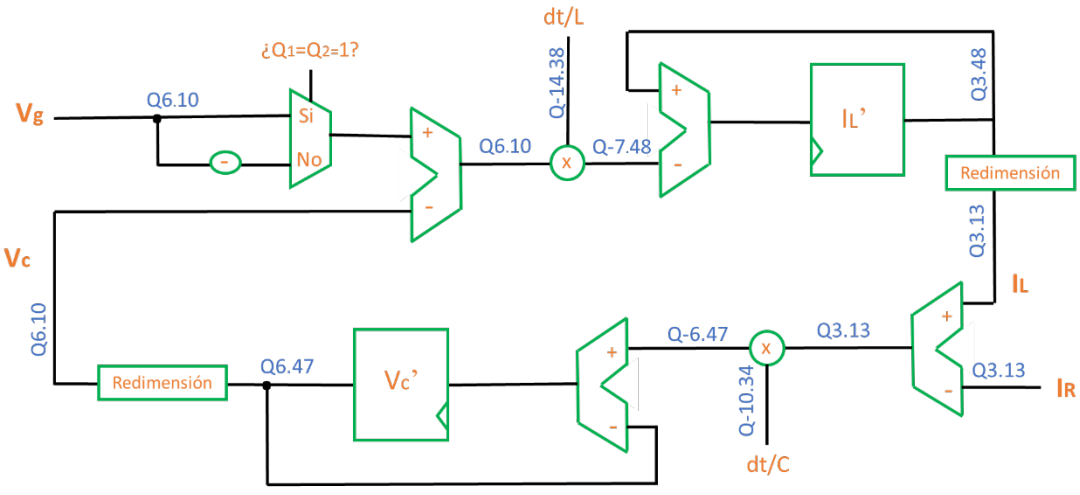

Figura 3-10. Hardware *full-bridge* del modelado en formato QX.Y

En la Figura 3.11 se pueden observar dos ejemplos de señales declaradas del tipo *sfixed*, las demás están declaradas de manera análoga. En este modelo las variables de

estado no se calculan directamente, sino que se realizan operaciones intermedias aplicando un redimensionado cuando es necesario. Las divisiones dt/L y dt/R, que corresponden en el código a dtL y dtC respectivamente, se precalculan para ahorrar la división en el código, lo que supone una mejora en la frecuencia, así como un número menor de recursos utilizados; esta simplificación en el modelo de tipo *real* no se lleva a cabo por no ser relevante. Las variables de estado se van actualizando en un proceso síncrono, pero necesitan finalmente ser redimensionadas, Figura 3.12.

```
--Valores auxilaries salida
      signal Ilaux      :     sfixed(3 downto -48); --Q3.48
      signal Voaux      :     sfixed(6 downto -47); --Q6.47
```

Figura 3-11. Ejemplo declaración de señales con librería *sfixed*

```
      process(Reset, Clk)
            begin
                  if (Reset = '1') then
                        Voaux <= (others => '0');
                        Ilaux <= (others => '0');
                  elsif (rising_edge(Clk)) then
                        Ilaux <= resize(Ilaux + deltaIL, ILaux);
                        Voaux <= resize(Voaux + deltaVo, Voaux);
                  end if;
      end process;

      VoauxRes <=  resize(Voaux, VoauxRes);

--Asigancion valores entrada y salida
      Il    <= resize(Ilaux, 3,-13);
      Vout  <= VoauxRes;
```

Figura 3-12. Valores de las variables de estado en coma fija

La resolución en el modelado con formato QX.Y no es un problema, ya que si por ejemplo se tiene una tensión de salida de 100 Voltios, una corriente que atraviesa la bobina de 5 Amperios y la corriente de la carga es de 4,9 Amperios, el valor de la tensión en el ciclo siguiente es:

$$v_C(k) = v_C(k-1) + \frac{\Delta t}{C} \cdot (i_L(k-1) - i_R(k-1)) = 100 + \frac{50 \cdot 10^9}{1 \cdot 10^{-4}} \cdot (5 - 4{,}9) = 100{,}00005\ V$$

Como el formato de la tensión del condensador en de Q6.47 el resultado del siguiente ciclo de reloj es representable; era lo esperado ya que se ha hecho un estudio previo del tamaño óptimo de cada señal. Este valor se guarda en 1 + 6 + 47 = 54 bits ($100{,}00005_{10}$= 01100100,00000000000001101000110110111000101110101100102); si estos bits no fueran suficientes para guardar la señal de salida la integración no se realizaría con precisión o incluso el valor sería irrepresentable. En cambio, la realimentación del modelo no necesita toda la información, por ello se redimensiona este valor reduciendo su tamaño hasta el formato Q6.10 en el cual la tensión de salida vale 100 Voltios debido a la aproximación. Las señales dtL y dtC representan valores decimales muy pequeños(formatos Q-14.38, desde el bit 15 al 38 decimal (1 – 14 + 38

= 25 bits) y Q-10.38, desde el bit 11 al 38 decimal (1 – 10 + 38 = 29 bits) respectivamente) ya que si dt, el valor que equivale al paso de integración, es de 50 ns, la inductancia de la bobina de 0,9 mH y el condensador de 0,1 mF, dtC = $5 \cdot 10^{-4}{}_{10}$ = $0{,}00000000010000011000100100110111010012$, del cual se extraen 27 bits de la parte decimal, es decir, $010000011000100100110111010012$. (El tamaño de cada señal se puede observar en la Figura 3.10).

La complejidad de este modelo es elevada, ya que, como se ha mencionado con anterioridad, hay que hacer un estudio por cada señal y operación para ajustar su tamaño al formato que le corresponde; gracias a esto, al sintetizar el modelo por una FPGA la relación entre velocidad y precisión puede ser óptima. Además, el modelado con esta notación presenta el inconveniente de tener que rediseñar el modelo, para ajustar los tamaños, si hay cambios en los valores reales durante la ejecución, evitando problemas de desbordamiento o de resolución insuficiente.

En el Anexo B se adjunta el código completo del modelo en coma fija.

3.2.3 Modelo en coma fija parametrizable del conmutador elevador

Como se explica en el modelado anterior, la aritmética en coma fija es sintetizable, pero presenta inconvenientes en cuanto se presentan modificaciones en los valores de los parámetros. En cambio, la coma fija parametrizable permite la adaptación del modelo sin tener que rediseñarlo; la adaptación consiste en cambiar la posición de la coma según sea necesario. Además, las características de la coma fija siguen estando presentes en esta aritmética, presentando frecuencias de funcionamiento muy altas, lo que permitirá emular el convertidor en tiempo real a la vez que se utilizan tiempos de integración muy bajos.

El tipo de datos deja de ser *sfixed* y pasa a ser *std_logic_vector* [15], que trabaja con vectores simples que pueden ser interpretados en complemento a dos y en principio no necesitan tener en cuenta los tamaños de la parte entera y fraccionaria. Cada señal necesita tener definida su escala, ya que será el número total en complemento a dos el que habría que dividir por la escala para obtener el resultado final; por ejemplo, si se tiene en binario $0010011_{2,}$ en decimal es 19_{10}, pero si la escala fuera 2 el valor de la señal sería 19/2 = 9,5. Esta escala se calcula restando al número total de bits el número entero mayor del logaritmo en base 2 del valor que se quiere representar. Gracias a la escala la coma se puede mover y acondicionar según los valores de los parámetros.

Con este modelo, el código de la fuente conmutada no requiere cambios. Como ya se ha mencionado, en este trabajo el código del modelo no se implementa ni se modifica, simplemente se adapta como periférico en el sistema integrado con el procesador tal y como se explicará más adelante. La gran ventaja de la coma fija parametrizable es que el modelo se sintetiza solo una vez y los únicos cambios necesarios son las configuraciones de los parámetros para que la simulación se adapte a las nuevas condiciones de contorno; para que la interfaz con el usuario sea fácil, los cálculos necesarios se pueden automatizar a través de una aplicación.

El objetivo principal del TFG no es el desarrollo de un modelo en coma fija parametrizable, sino dotarlo de comunicación con el microprocesador desarrollado. En

particular, este TFG consiste en añadir capacidad de comunicación a un modelo de un convertidor elevador o *boost* con pérdidas cuyo esquemático se adjunta en la Figura 3-13. El modelo *full-brige* que se ha mostrado anteriormente ha servido para explicar la metodología de modelado de un convertidor conmutado. Una vez aprendida la metodología, el modelado de otro convertidor se realiza de forma similar.

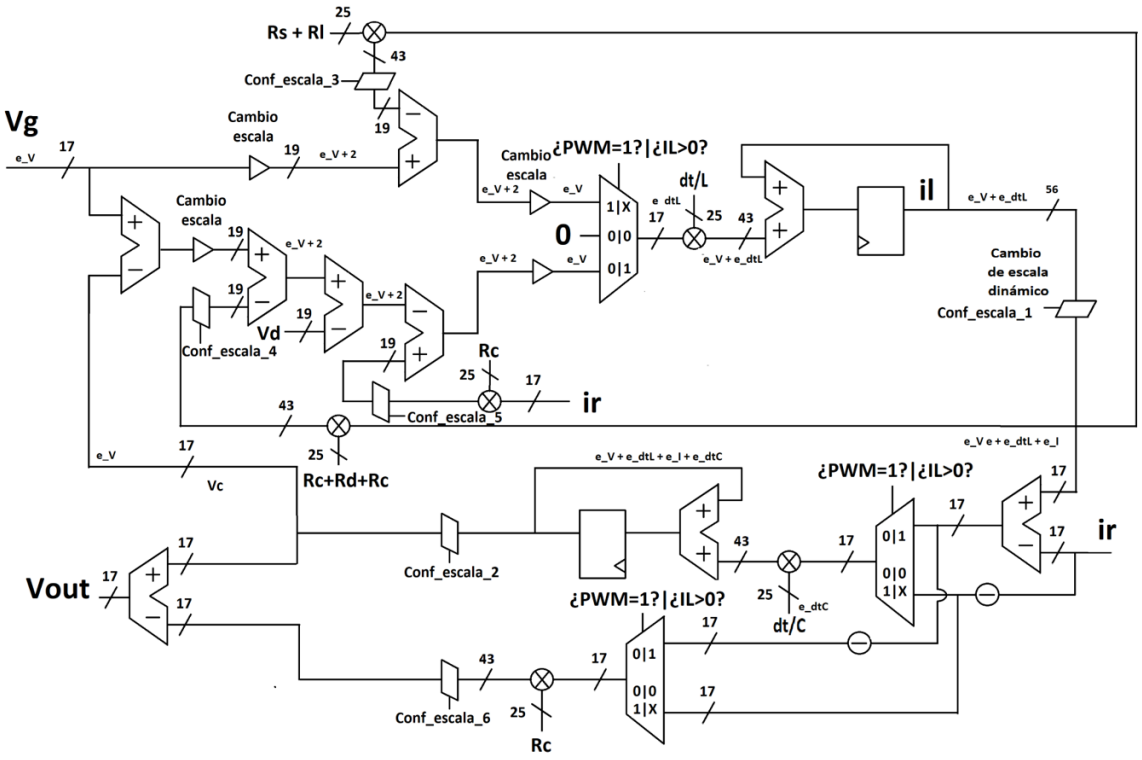

Figura 3-13. [15] Esquemático del *boost* implementado en coma fija parametrizable

4 Integración del modelo HIL en un sistema embebido

Como ya se ha comentado, para poder utilizar las grandes ventajas que presenta el modelo de una planta implementada con la aritmética en coma fija parametrizable, es necesario dotar a este modelo de la capacidad de comunicación con el usuario para que se pueda configurar. Para que esta comunicación se pueda llevar a cabo, hay que integrar el modelo en una FPGA.

La FPGA utilizada es la Zynq-7000 de Xilinx, que además de tener lógica programable que facilita el acceso a los usuarios, contiene un microprocesador de silicio, ARM Cortex^{TM}-A9 de doble núcleo. Aunque la utilización de un microprocesador no sería necesaria, es de gran utilidad, ya que da versatilidad al sistema en cuanto a posibles mejoras futuras.

En la Figura 4-1 se ilustra el sistema integrado completo, en el que el microprocesador se conecta con todos los periféricos a través del bus principal. Como se puede ver, el encargado de que se produzca la comunicación entre el sistema y el usuario es el periférico UART (*Universal Asynchronous Receiver-Transmitter)*. Además, hay otro periférico integrado denominado *Timer*, que es el encargado de que las comunicaciones se produzcan de manera ordenada evitando que se den desincronizaciones entre el usuario y la FPGA. Por último, se encuentra el periférico del modelo implementado que es controlado por un PWM (*Pulse Width Modulation)* externo y sus señales de salida están en digital, por eso se conectan a unos DACs convirtiéndolas en analógicas; este modelo de la planta se comunica con el microprocesador gracias al banco de registros que se le asigna.

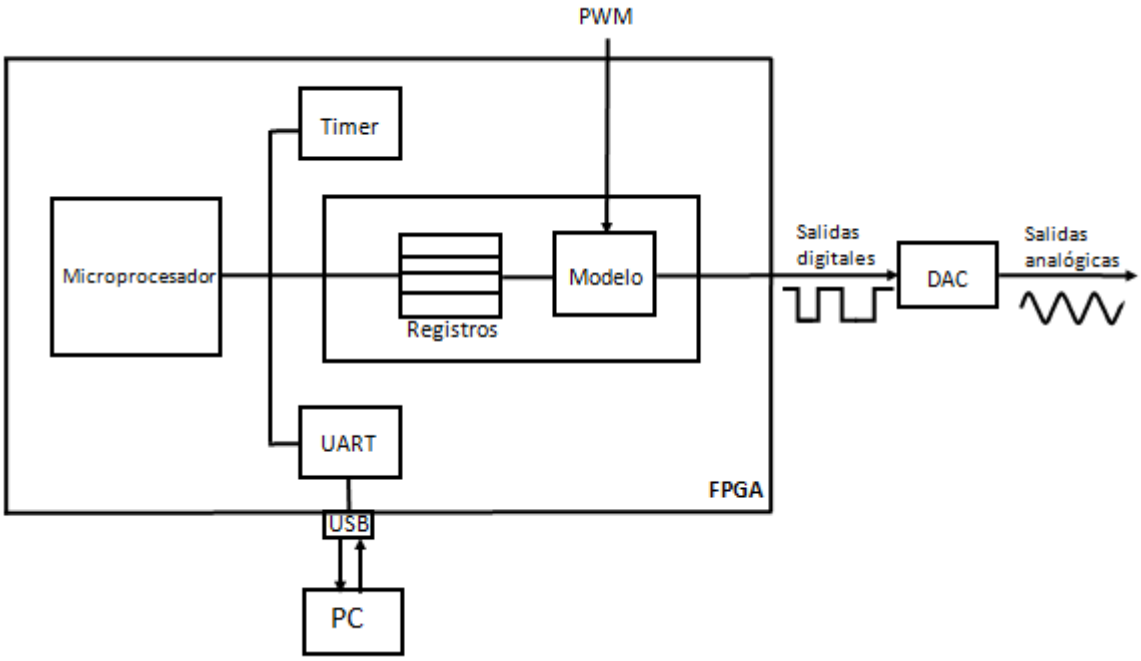

Figura 4-1. Sistema integrado completo

4.1 Arquitectura

Para conseguir la comunicación del modelo HIL con el exterior, hay que integrar el modelo en un periférico del microprocesador y que así pueda ser configurado; esta configuración la realiza el usuario a través del ordenador y debe llegar al sistema como entrada a la FPGA mediante el puerto USB. La comunicación entre el microprocesador y el modelo de la planta se realiza con registros de entrada y salida, a los que se les asignan las señales que escriben o leen los datos de los registros.

Las variables de estado del conmutador elevador son salidas digitales que se conectan a los DACs que las convierten en analógicas, estos conversores son necesarios ya que la planta se ha digitalizado por cuestiones favorables a la simulación del sistema completo, pero para poder sustituir la planta del convertidor por el modelo, las señales sensadas a la salida de este deben ser analógicas ya que es lo esperado por cualquier control de una planta.

Para que el modelo de la planta y el microprocesador se puedan comunicar tienen que tener la misma frecuencia de conmutación. La frecuencia máxima admitida por el *boost* es de 28 MHz como se puede observar en el camino crítico de la Figura 4.2 (el camino crítico de un diseño es el máximo retraso que hay desde la salida de un Flip-Flop hasta la entrada de otro) y la frecuencia por defecto del procesador es de 100 MHz, la cual ha tenido que modificarse.

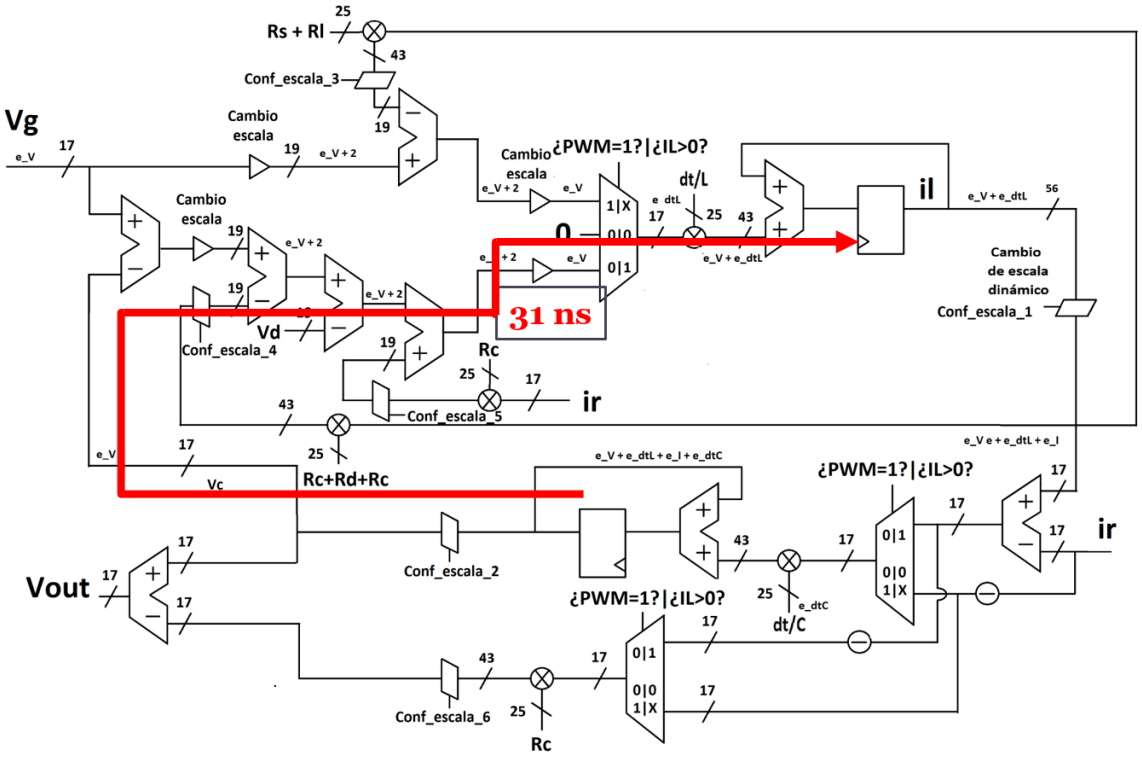

Figura 4-2. Camino crítico coma fija parametrizable

Tanto la UART como el Timer, ambos periféricos del microprocesador empotrado se pueden observar en la Figura 4-1, se configuran en software. Por un lado, la UART es la que hace posible la comunicación entre usuario y FPGA; pero es el Timer el que permite que la comunicación sea correcta, se encarga de que las tramas que se envían sean correctas y evita que la FPGA y la aplicación del usuario queden desincronizadas.

4.2 Comunicación entre el modelo y el exterior

Como se ha mencionado anteriormente, la comunicación entre procesador y los periféricos, UART y Timer, se realiza mediante la gestión de interrupciones, un mecanismo que permite interrumpir la ejecución del programa ejecutado por el microprocesador, disminuyendo así la carga de este. En la Figura 4.3, se ilustra la gestión de interrupciones; cuando el microprocesador está ejecuta el programa principal (I_1, $I_{2...}$) y detecta una interrupción (I_i de la Figura 4.3) el programa principal queda suspendido quedando libre mientras se ejecuta la rutina de servicio de interrupción; esta termina cuando llega a la instrucción de retorno de interrupción (I_m), con lo que el microprocesador sigue ejecutando el programa que había sido interrumpido (I_{i+1}).

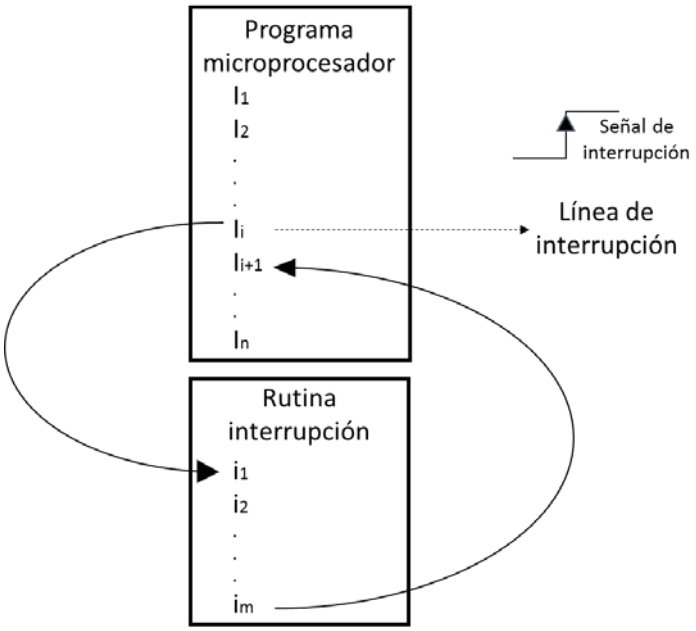

Figura 4-3. Gestión de interrupciones

La comunicación del microprocesador con la UART consiste en el envío de tramas, siendo las interrupciones las encargadas de notificar al procesador que se ha recibido un byte, de esta forma el microprocesador no tiene que estar preguntado continuamente hasta recibir el principio de trama. Cuando se empieza a recibir una trama se detecta una interrupción y el programa principal permanece suspendido hasta que se termine la transmisión. Para que puedan producirse interrupciones es necesario configurarlas, como se muestra en el Anexo C.

Tanto el Timer como la UART tienen que habilitarse, para ello se tienen que configurar en lenguaje C a través del programa SDK de Eclipse (La configuración de la UART se adjunta en el Anexo D y la del Timer en el Anexo E). Se habilitan ambos periféricos mediante su dirección de memoria, disponible en el fichero "xparameters.h", el cual contiene los parámetros del sistema para el entorno del controlador de dispositivo Xilinx. Como ya se ha dicho, la comunicación con el microprocesador se da gracias a las interrupciones por lo que ambos periféricos deben conectarse a este subsistema.

La UART, el periférico que activa las interrupciones al recibir un byte, tiene que tener configurados sus controladores para que los datos que se reciban puedan ser llamados desde el contexto de interrupción; para depositar los datos recibidos utiliza un buffer de recepción.

4.2.1 Timer

Aunque es la UART la que a través del puerto USB comunica al usuario con la FPGA, el Timer tiene como función imprescindible cerciorar que la comunicación se produce correctamente. Cuando se recibe un byte, se considera principio de trama, pero por algún problema externo puede que no lo sea; si esto ocurre en ausencia del Timer la aplicación del usuario y la FPGA se desincronizan. El Timer, para que la FPGA y el usuario estén sincronizados permanentemente, fija un *timeout*, para que si el tiempo entre bytes lo supera se deseche la trama. Como la velocidad de transmisión es de 115200 bps y se envían 10 bits (1 byte tiene 8 bits más los 2 de inicio y final de trama) el tiempo entre bytes es aproximadamente de 86,80 µs; el *timeout* es de 1 ms para dejar un margen más que suficiente por posibles retardos. Por lo tanto, si se reciben dos bytes consecutivos en un tiempo superior a 1 ms el segundo byte se considera principio de trama; es decir, si un byte no se recibe cuando se espera, el Timer evita la desincronización de la aplicación del usuario con la placa recuperando información. Para mayor claridad se ilustra en la Figura 4.4, además del pseudocódigo en la Figura 4-5 (El código se adjunta en el Anexo F).

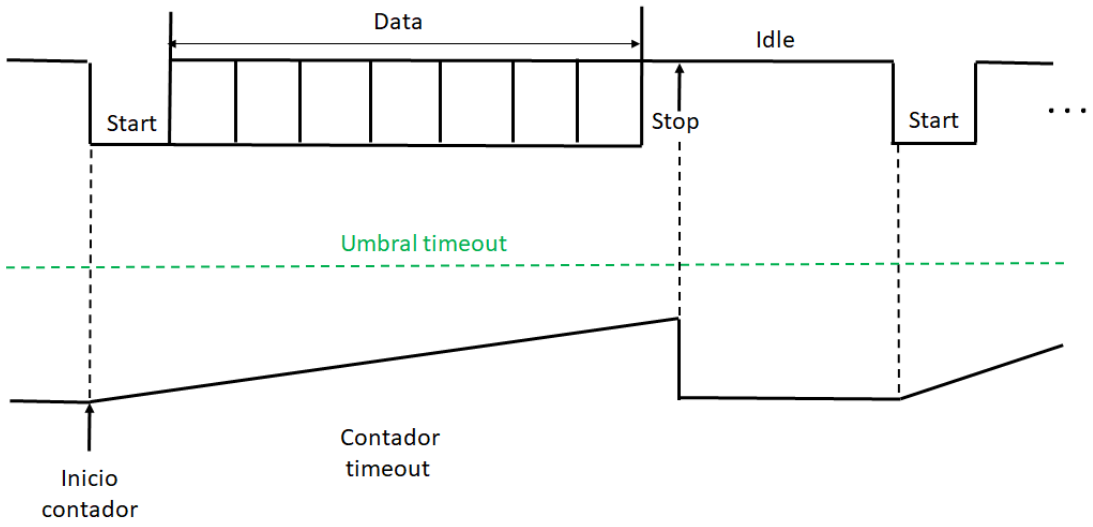

Figura 4-4. Envío de tramas

```
bucle infinito
      se reciben los 6 bytes esperado
            Stop Timer
            se sale del bucle

      se reciben menos de 6 bytes
            bandera del timer desactivada
                  Start Timer

            banderas de timer y timeout activadas
                  se descarta trama
                  se sale del bucle
```

Figura 4-5. Pseudocódigo de la función del Timer

Se tiene que comprobar que las tramas recibidas son válidas, para ello se cuenta el número de bytes recibidos y el Timer se encarga de comprobar que la recepción se ha hecho en el tiempo esperado. El número de bytes por trama es 6, como se puede observar en la Figura 4-6, siendo el primero el número de registro, así el programa detecta que configuración se está modificando, es decir, indica cuál es el registro que se va a actualizar y por ende que señal, ya que cada señal esta asignada con su registro correspondiente; los cuatro bytes siguientes se concatenan formando los datos que van a escribirse en el registro indicado por el byte de principio de trama; el último es el checksum, función hash para detectar cambios accidentales en la trama y así proteger la integridad, es decir, comprueba que la trama está bien formada.

Numero registro	dato [0]	dato [1]	dato [2]	dato [3]	Checksum

Figura 4-6. Trama

Para poder considerar válida a una trama, el checksum indicado por el último byte tiene que coincidir con el checksum calculado por la FPGA. El checksum se calcula haciendo la operación XOR de la información de los cinco primero bytes, es decir, checksum = num_registro $\oplus$ dato[0] $\oplus$ dato[1] $\oplus$ dato[2] $\oplus$ dato[3].

$$DirRW = Dir_{base\ perif} + num\_registro \qquad (4.1)$$

Al periférico creado del modelo de la planta se le han asignado 32 registros de 32 bits, aunque no todos ellos se utilizan, se deja un margen por posibles modificaciones futuras. En la Tabla 4-1 se observa la configuración de todos los registros (se han configurado con pérdidas), es decir, la señal o señales que se le han asignado a cada uno, por lo que el usuario a través de la aplicación, enviará los datos de las señales que quiere actualizar y gracias a estas asignaciones las señales quedarán actualizadas.

Tabla 4-1. Asignación de los registros

	Reg \ Bits	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Write/Read	0				Desp_RL_RD_RC																dtL													
Write/Read	1					Desp_RL_RS															dtC													
Write/Read	2						Desp_RC_2						Desp_RC_1												Vc_init								
Write/Read	3						Desp_Vout						Desp_IL												IL_init								
Write/Read	4																							Vd								
Write/Read	5				Desp_GOUT															Rl_Rs												
Write/Read	6																			Rl_Rd_Rc												
Write/Read	7																			Rc												
Write/Read	8																							Vg								
Write/Read	9																			Gout												
Write/Read	10																																Load
Write/Read	11				Desp_DACVout															multDacVout													
Write/Read	12																							offset_DACVout								
Write/Read	13				Desp_DACVc															multDacVc												
Write/Read	14																							offset_DACVc								
Write/Read	15				Desp_DACIL															multDacIL												
Write/Read	16																							offset_DACIL								
Read	17																							Vc								
Read	18																							Vout								
Read	19																			IL												

En cada emulación, a los registros de escritura del periférico se les pasan unos valores para que así puedan llegar a las señales que tienen asignadas estos registros y actualizarse. A continuación, se entra más en detalle sobre esta comunicación. Cabe destacar que al programa ejecutado no se le dota de inteligencia para recibir parámetros aislados y componer los registros de configuración del modelo, si no que los datos que envía el usuario desde la aplicación se agrupan de tal forma que al programa ejecutado en el procesador de la FPGA ya le llega un dato completo para guardar en un registro del periférico del modelo.

Como se ha mencionado anteriormente, la comunicación entre el microprocesador y el modelo de la planta se hace a través del banco de registros del módulo de E/S del periférico. Este sistema facilita las operaciones de entrada y salida entre el *boost* y el microprocesador. Para poder realizar la comunicación se necesita un hardware acompañado de un software específico, los cuales se han ido comentando a lo largo del trabajo. Es una comunicación diferente a la de gestión de interrupciones utilizada en los dispositivos UART y Timer de la que se habla anteriormente.

El módulo de E/S mapeado en memoria utiliza un solo bus para dar acceso al banco de registros y a las direcciones de memoria; además, las instrucciones del microprocesador para acceder a los registros, que contienen los datos de las señales, son también utilizadas para acceder a memoria.

El bus del sistema es el encargado de comunicar a la planta que se están transmitiendo datos. La comunicación es realizada a través de instrucciones que leen y escriben en registros. El microprocesador de silicio interpreta al banco de registros como un conjunto de espacios de memoria, pero lo que ocurre realmente es que los registros están mapeados a sus direcciones de memoria correspondientes.

Cuando el usuario quiera configurar el modelo, cambiará el valor de los parámetros dinámicos y los mandará; lo que da lugar a la comunicación de la que se está hablando en esta sección. Esta transmisión consta de dos procesos, el de sincronización que se encarga de comprobar que el periférico está preparado para recibir información y el que de verdad envía el dato.

En la aplicación para el usuario se han asignado unos registros de lectura y otros de escritura, que sirven para operaciones de entrada y salida respectivamente. En las operaciones de entrada es el microprocesador el que quiere leer datos que han sido enviados por el convertidor conmutado y están disponibles en el banco de registros del módulo de E/S, tan solo se han asignado 3 registros de lectura, a los que se les asignarán los datos de la corriente que atraviesa la bobina y los de las tensiones de salida y en bornes del condensador. Las de salida son las encargadas de transmitir las configuraciones del usuario, es decir, es el procesador ARM Cortex™-A9 el que escribe los datos en los registros de escritura y luego estos se envían al modelo.

En las instrucciones del microprocesador hay que especificar la dirección de memoria que corresponde a cada registro que se conecta con el periférico *boost*. Esas direcciones se envían por el bus y el módulo de entrada y salida es el que se encarga de enviar la información actualizada de los registros hasta el modelo, si se trata de una operación de escritura, quedando el conmutador elevador configurado con las especificaciones del usuario; si se trata de una operación de salida los datos del banco de registros se envían por el bus del sistema hasta llegar al procesador.

Gracias al diseño implementado en coma fija parametrizable, una única síntesis sirve para realizar todas las emulaciones que se quieran mientras esta se ejecuta. Las emulaciones sirven para configurar el conmutador elevador con los parámetros dinámicos que el usuario elija, siempre y cuando estén dentro de los rangos posibles.

Ya se ha comentado que para poder aprovechar las características de la aritmética del tipo coma fija parametrizable es esencial poder poner al usuario en contacto con el modelo de la planta para realizar así diferentes emulaciones, las cuales se realizan en tiempo real y con un paso de integración muy pequeño, lo que dota al sistema de una elevada resolución.

Se han mostrado los datos que se han escrito en los registros para que el *boost* pueda configurar las señales que van asignadas a ellos para una emulación, en la siguiente sección se detalla más sobre los datos que se escriben en los registros de cualquier emulación.

5 Integración, pruebas y resultados

Se ha mencionado anteriormente que, para añadir comunicación a un diseño VHDL, no es imprescindible utilizar un procesador, pero dota al sistema de gran versatilidad de cara a ampliaciones futuras. El sistema final embebido se ha configurado con la herramienta Vivado 2015.4 y se puede observar en la Figura 5.1. Para extraer la información que aporta el conmutador elevador en coma fija parametrizable se conectan las salidas a través de unos DACs, ya que las señales están en digital y para observar sus comportamientos las necesitamos en analógico.

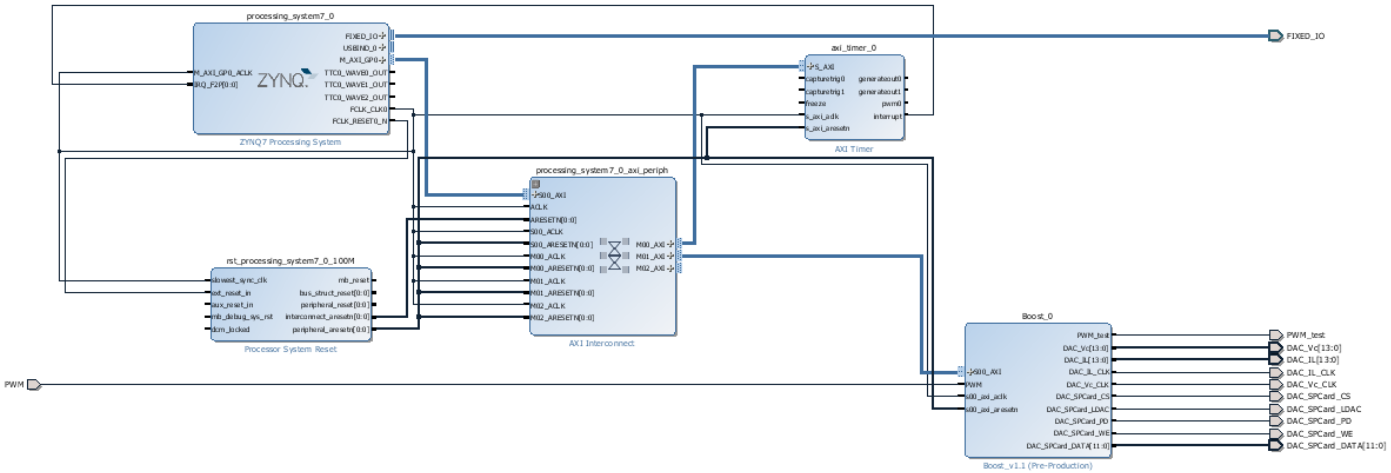

Figura 5-1. Sistema integrado completo en Vivado

Como se acaba de comentar, las señales que contienen la información de las variables de estado (corriente que atraviesa la bobina y tensión que cae en el condensador), tienen que conectarse a los convertidores de señal digital a analógica. Uno de los DAC está integrado en la FPGA y el otro es externo a ella. Se quiere ver estas salidas en un osciloscopio, para ello se necesita un regulador en lazo abierto. Con un generador de señal, modelo WW2571A, se crea el ciclo PWM (hace de regulador en lazo abierto) una frecuencia de conmutación de 100 KHz al 50%.

En el capítulo 4.1, los conversores de las señales de digital a analógico son fundamentales porque, aunque se haya digitalizado la planta para agilizar la simulación del sistema embebido, las salidas de la planta real son analógicas, por lo tanto las señales sensadas a la salida deben serlo también.

Una vez se tienen las señales convertidas a analógico, se conectarían a un control en lazo cerrado, lo cual no es objeto de este trabajo. Estas salidas se conectan a las sondas de un osciloscopio, MSO-X 3014A, para poder observar el comportamiento de las señales y compararlas con las diferentes emulaciones. Como se explica en el capítulo anterior, hay que volver a configurar el modelo cambiando los parámetros

dinámicos si se quiere conseguir una emulación nueva, los fijos no pueden cambiarse mientras se está ejecutando la simulación (Tabla 5-1).

Tabla 5-1. Valores de los parámetros fijos correspondientes a la Figura 5-2

Parámetros fijos	**Valores**	
L	1 mH	
C	0,1 µF	
R_L	0,6965 Ω	
R_S	0,3 Ω	
R_D	0,01 Ω	
R_C	0,001 Ω	
V_D	1,3 V	
I_{max}	3 A	5 A
V_{max}	64 V	500 V

Se van a presentar cuatro emulaciones diferentes, cuyas configuraciones de los parámetros dinámicos se ilustran en la Tabla 5-2. Los valores que se muestran en la Tabla 5-2 se han elegido para ilustrar los ejemplo, pero al ser parametrizable, se pueden hacer todas las simulaciones que se quiera sin tener que resintetizar el modelo. La resistencia mínima que admite el modelo es de 21,3. Ya se ha explicado que, para poder observar los comportamientos, las señales han de conectarse a un osciloscopio.

Además de estos valores dinámicos, se puede elegir la salida máxima que representan los DACs, siendo este valor el equivalente a los 5 (voltios o amperios, dependiendo de la señal de salía) que es la salida máxima que los conversores son capaces de representar. El valor máximo elegido para cada DAC en las emulaciones se muestra en la Tabla 5-3. Es importante destacar que esta salida de los DACs representan la tensión de salida que cualquier convertidor de potencia ofrece para poder cerrar un lazo de control. Para dicho sensado, los convertidores tienen divisores resistivos y amplificadores operacionales para reducir y adaptar la tensión a los márgenes de sensado. Por tanto, los DACs del modelo HIL generan tensiones equivalentes a las salidas adaptadas de un convertidor real.

Las señales que se registran en la pantalla del osciloscopio (Figura 5.2) son productos de las emulaciones realizadas con los valores de la Tabla 5-2 y la Tabla 5-3. La curva naranja-arriba representa la tensión en bornes del condensador y la azul-abajo la corriente que atraviesa la bobina. La forma de la curva de tensión tiene más ruido porque es la que se conecta al DAC integrado en la FPGA, que es de peor calidad que el externo que utiliza la señal de corriente.

En los cuatro casos se puede observar que ambas señales son sistemas de segundo orden subamortiguados, el factor de amortiguamiento es menor que 1, es decir presentan oscilaciones decrecientes hasta llegar al estado estacionario.

Si se comparan las gráficas **a)** y **b)** de la Figura 5.2 que tienen ambas la misma tensión de entrada y los mismos valores máximos representados por los DACs pero **a)**

una resistencia cercana a la mínima y **b)** de 200 Ω. Las señales de tensión de ambas emulaciones se comportan de manera similar, tienen una sobreoscilación acentuada y el régimen permanente es prácticamente el mismo pese a la gran diferencia del valor de la resistencia; bastante elevado en comparación con su valor al principio del ciclo. Las señales de corriente, aunque ambas tienen una sobreoscilación marcada la de la gráfica **b)** en régimen permanente queda muy cerca de su valor al principio del ciclo de conmutación.

Comparando **a)** y **c)** de la Figura 5.2 , ambas con la misma resistencia cercana a la mínima pero **c)** con una tensión de entrada menor (muy pequeña) y los mismos valores máximos representados por los DACs. La sobreoscilación de las dos señales en la gráfica **c)** es menor, lo que implica que sus señales llegan antes al estado estacionario, además, tienen un valor menor al final del ciclo.

Finalmente, si se comparan **b)** y **d)**, no tiene sentido fijarse en los valores de la tensión de entrada y la resistencia para compararlos, ya que los valores que representan la salida máxima de los DACs en este caso son muy diferentes, en la grafica **d)** estos valores multiplican por 10 a los de **c)**. Se hace una comparación porque la corriente que atraviesa la bobina en ambos casos, cuando llega al régimen permanente se aproxima mucho a la corriente al principio del ciclo, se puede concluir que es por el valor bajo de la tensión de entrada con respecto al valor máximo que representa la salida del DAC.

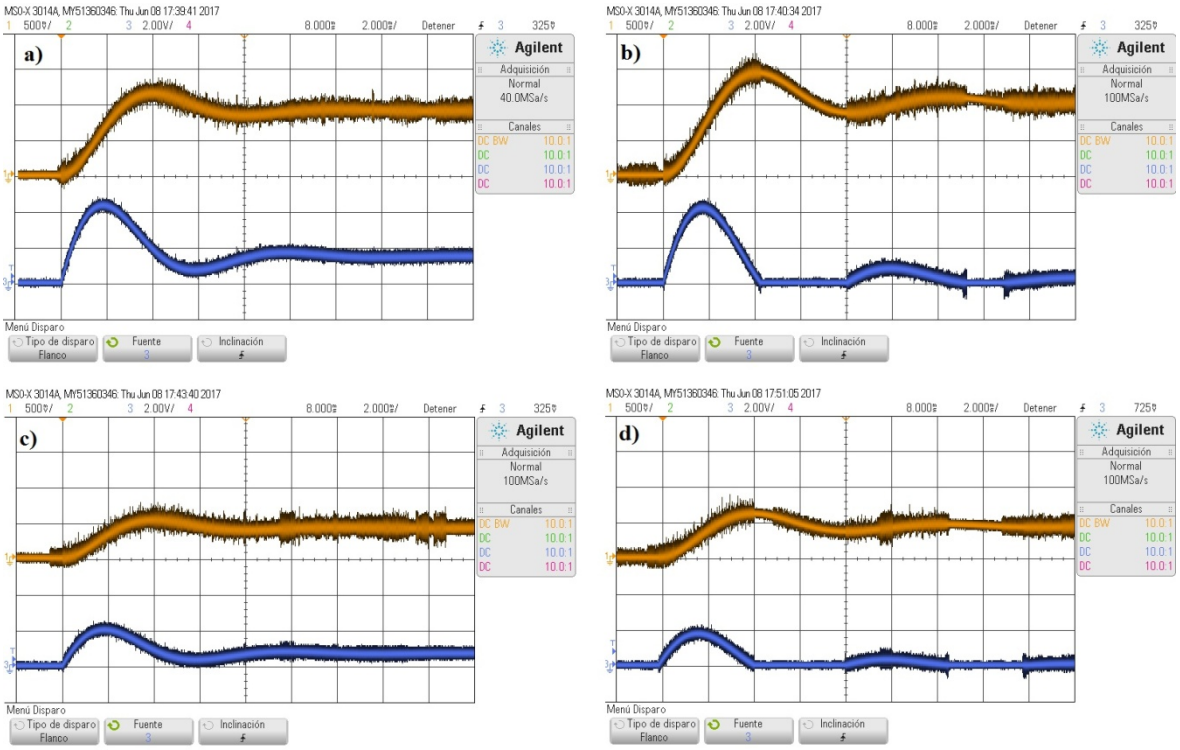

Figura 5-2. Variables de estado con distintos parámetros dinámicos (curva naranja-arriba tensión del condensador, curva azul-abajo corriente de la bobina)

Tabla 5-2. Valores de los parámetros dinámicos

Parámetros dinámicos	**Figura 5-2**			
	a	b	c	d
R (Ω)	22	200	22	500
V_{IN} (V)	10	10	5	40

Tabla 5-3. Valores que representan el valor máximo de los DACs

Valor máximo DAC	**Figura 5-2**			
	a	b	c	d
V_c (V)	60	60	60	600
I_L (A)	5	5	5	50

Después de haber integrado el modelo HIL con el modelo de la planta en un sistema embebido con un microprocesador y realizar todas las comunicaciones necesarias con sus periféricos integrados, era necesario dotar al periférico del conmutador elevador de comunicación con el usuario para que este pudiera configurarlo y poder así realizar las emulaciones convenientes en tiempo real y con un tiempo de integración bajo gracias al modelado en coma fija parametrizable. Finalmente, se ha comprobado que sin haber resintetizado por el puerto serie, se pueden apreciar transitorios del arranque hasta el régimen permanente.

6 Conclusiones y trabajo futuro

6.1 Conclusiones

Cualquier convertidor de potencia conmutado necesita un controlador que la regule; los controladores cada vez más se implementan en digital por las ventajas que presenta. La depuración de este control es esencial porque si tiene fallos de implementación al conectarlo a la planta real puede estropearla debido a una subida de corriente, que puede originar daños materiales que desembocan en económicos además de personales.

No solo es indispensable la simulación del regulador por separado, sino la comprobación con un modelo de planta para asegurarse de que la planta real se va a controlar sin ningún riesgo. Aunque existen herramientas capaces de conseguir modelos matemáticos de la planta óptimos, su implementación puede contener algún error. Por eso se han buscado técnicas de depuración que aseguren el correcto funcionamiento del sistema completo.

Las simulaciones del sistema tienen que ser mixtas, por tener el control digital y la planta analógica. Existen en el mercado diferentes simulaciones mixtas, pero tienen el gran inconveniente de que el tiempo de simulación es elevado.

Para solventar el problema del tiempo de simulación, se ha utilizado la técnica HIL, que consiste en digitalizar la planta implementándola en hardware, en una FPGA, que permite reducir el tiempo de integración al máximo dando lugar a simulaciones más precisas. Gracias a este diseño la simulación se simplifica porque puede producirse como un sistema único.

Se han desarrollado las ecuaciones de un convertidor de potencia para transcribirlas en VHDL e integrar el modelo en un periférico de un sistema embebido con procesador. Para ello se han desarrollado dos modelados, uno tipo *real*, que presenta el vital problema de no poder sintetizarse pero es de fácil implementación y es muy preciso, por lo que este diseño es de utilidad para compararlo con otras aritméticas; y otro en coma fija, que ha requerido un mayor esfuerzo de diseño al tener que fijar el formato de cada señal, además no es universal, un cambio en las variables implica tener que rediseñar el modelo. Por eso se ha utilizado la aritmética en coma fija parametrizable que permite diferentes emulaciones cambiando los parámetros dinámicos sin tener que resintetizar el modelo. El inconveniente que tiene dicha aritmética es que debe configurarse en tiempo de ejecución para poder ser utilizada. Para facilitar dicha tarea, se ha dotado al sistema de comunicación serie para poder recibir la configuración desde un ordenador.

Se ha dotado, por tanto, al sistema de comunicación para que se puedan ejecutar las diferentes emulaciones gracias las configuraciones realizadas por el usuario. Para cerciorar esa comunicación se han conectado las señales de salida a un osciloscopio en el que con diferentes configuraciones se han visto distintos transitorios del arranque hasta llegar al régimen permanente de estas señales.

6.2 Trabajo futuro

En este Trabajo de Fin de Grado se ha integrado el modelo de un convertidor de potencia en un periférico de un sistema embebido con microprocesador y se le ha aportado comunicación con el usuario para que este lo pueda configurar. Con ello aparecen líneas de trabajo futuro, de mejora y nuevas aportaciones:

- Mejorar la comunicación entre el usuario y el modelo de la planta para facilitar su configuración desarrollando una aplicación en el ordenador que sea capaz de mandar datos, es decir, mandar los valores de las variables de estado.
- Aprovechando la capacidad de mejora del sistema gracias al uso de un procesador, se propone implementar un sistema de monitorización de alarmas para que indique si algún parámetro (tensión, corriente, frecuencia de conmutación, etc.) se aleja de los valores esperados.
- También se propone la implementación de un osciloscopio digital en el ordenador cuyos datos serían enviados desde el procesador de la FPGA hasta el ordenador de forma periódica, y comprobar en él los comportamientos de las señales de salida del modelo del conmutador de potencia.

Referencias

[1] A. de Castro, *Aplicación del Control Digital Basado en Hardware Específico para Convertidores de Potencia Conmutados*. PhD thesis, Universidad Politécnica de Madrid, 2003.

[2] Ned Mohan, Tore M. Undeland, William P.Robbins. Electrónica de Potencia. Convertidores, aplicaciones y diseño. 3º Edición – 2009.

[3] Matlab, "www.mathworks.com," 2013.

[4] A. Prodic and D. Maksimovic, "Mixed-signal simulation of digitally controlled switching converters," in Proc. IEEE Workshop Comput. Power Electron., Mayaguez, PR, USA, Jun. 2002, pp. 100–105

[5] L. Barragan, I. Urriza, D. Navarro, J. Artigas, J. Acero, and J. Burdio, "Comparing simulation alternatives of FPGA-based controllers for switching converters," in Proc. IEEE Int. Symp. Ind. Electron. (ISIE), Boulder, CO, USA, Jun. 2007, pp. 419–424.

[6] I. Urriza, L. Barragan, J. Artigas, J. Acero, D. Navarro, and J. Burdio, "Using mixedsignal simulation to design a digital power measurement system for induction heating home appliances," in Industrial Electronics, 2007. ISIE 2007. IEEE International Symposium on, pp. 1447-1451, jun. 2007

[7] A. Procdic and D. Maksimovic "Mixed-signal simulation of digitally controlled switching converters". In Computers in Power Electronics, 2002. Proceedings. 2002 IEEE Workshop on pp. Junio 2002.

[8] P. Zumel, M. García-Valderas, A. Lázaro, C. López-Ongil, and A. Barrado, "Cosimulation PSIM-ModelSim oriented to digitally controlled switching power converters," in Proc. IEEE 12th Workshop Control Model. Power Electron. (COMPEL), Jun. 2010, pp. 1–7

[9] S. Karimi, P. Poure, and S. Saadate, "An HIL-based reconfigurable platform for design, implementation, and verification of electrical system digital controllers," IEEE Trans. Ind. Electron., vol. 57, no. 4, pp. 1226–1236, Apr. 2010.

[10] M. Matar and R. Iravani, "FPGA implementation of the power electronic converter model for real-time simulation of electromagnetic transients," IEEE Trans. Power Del., vol. 25, no. 2, pp. 852–860, Apr. 2010.

[11] Y. Chen and V. Dinavahi, "Digital hardware emulation of universal machine and universal line models for real-time electromagnetic transient simulation," IEEE Trans. Ind. Electron., vol. 59, no. 2, pp. 1300–1309, Feb. 2012.

[12] C. Dufour, V. Lapointe, J. Belanger, and S. Abourida, "Hardware-in-the-loop closedloop experiments with an fpga-based permanent magnet synchronous motor drive

system and a rapidly prototyped controller," in Industrial Electronics, 2008. ISIE 2008. IEEE International Symposium on, pp. 2152-2158, jul. 2008.

[13] M. Matar and R. Iravani, "Fpga implementation of the power electronic converter model for real-time simulation of electromagnetic transients," Power Delivery, IEEE Transactions on, vol. 25, pp. 852-860, apr. 2010.

[14] A. Sánchez González, *Aportaciones mediante implementación basada en sistemas embebidos al control digital de convertidores conmutados*, Tesis Doctoral, jun 2013.

[15] V. Pinazo, *Implementación de un modelo HIL en coma fija parametrizable con pérdidas eléctricas*, jun 2017.

Glosario

DAC	*Digital to Analog Converter*
FPGA	*Field-Programmable Gate Array*
HDL	*Hardware Description Languaje*
HIL	*Hardware In the Loop*
PWM	*Pulse Width Modulation*
UART	*Universal Asynchronous Receiver-Transmitter*
VHDL	*VHSIC Hardware Description Languaje*

Anexos

A Código del full-bridge en aritmética tipo real

```
----------------------------------------------------------------------------
---------
-- Company:
-- Author:  Beatriz Camara de la Peña
--
-- Create Date:    09/27/2016
-- Module Name:    proyecto - Behavioral
-- Description:    aritmetica tipo real
-- Revision 0.01 - File Created
----------------------------------------------------------------------------
---------
library IEEE;
use IEEE.std_logic_1164.all;
use IEEE.std_logic_arith.all;
use IEEE.std_logic_signed.all;
use IEEE.math_real.all;
use IEEE.numeric_std.all;

entity proyecto is
      port (
                  Clk         : IN std_logic;
                  Reset       : IN std_logic;
                  Vg          : IN real;
                  Ir          : IN real;
                  Q1          : IN std_logic;
                  Q2          : IN std_logic;
                  Q3          : IN std_logic;
                  Q4          : IN std_logic;
                  Il          : OUT real;
                  Vout        : OUT real
            );
end proyecto;

architecture Behavioral of proyecto is

      --valores delta
      signal VoDelta    :     real;
      signal IDelta           :     real;

      --valores auxiliares salida
      signal Ilaux      :     real := 0.0;
      signal Voaux      :     real := 0.0;

      --constantes
      signal C    :     real;
      signal L    :     real;
      signal dt   :     real;

begin

      C     <= 1.0e-4;
      L     <= 9.0e-4;
      dt    <= 50.0e-9;
```

```
	process (Reset, Clk)
	begin
		if (Reset = '1') then
			Voaux <= 0.0;
			Ilaux <= 0.0;
		elsif (rising_edge(Clk)) then
			Voaux <= Voaux + IDelta*dt/C;
			Ilaux <= Ilaux + VoDelta*dt/L;
		end if;
	end process;

	--Calculo de las operaciones intermedias
	IDelta		<= (Ilaux-Ir);
	VoDelta	<= (Vg-Voaux) when (Q1='1' and Q2='1') else (-Vg-
Voaux);

	--Asigancion valores entrada y salida
	Il	<=	Ilaux;
	Vout	<= Voaux;

end Behavioral;
```

B Código del full-bridge en coma fija

```
----------------------------------------------------------------------------
---------
-- Company:
-- Author:  Beatriz Camara de la Peña
--
-- Create Date:    15:19:52 11/14/2016
-- Module Name:    sfixed - Behavioral
-- Description:    aritmetica sfixed
----------------------------------------------------------------------------
---------
library WORK, IEEE, IEEE_proposed;
use IEEE.std_logic_1164.all;
use IEEE.numeric_std.all;
use IEEE_proposed.fixed_float_types.all;
use IEEE_proposed.fixed_pkg.all;
use IEEE.math_real.all;

entity sfixed_modulo is
      Port (
            Clk         : IN std_logic;
            Reset       : IN std_logic;
            Vg          : IN sfixed(6 downto -10);
            Ir          : IN sfixed(3 downto -13);
            Q1          : IN std_logic;
            Q2          : IN std_logic;
            Q3          : IN std_logic;
            Q4          : IN std_logic;
            Il          : OUT sfixed(3 downto -13);
            Vout        : OUT sfixed(6 downto -10)
      );
end sfixed_modulo;

architecture Behavioral of sfixed_modulo is

      --Valores delta
      signal VoDelta    :     sfixed(7 downto -10); --Q7.10
      signal IDelta     :     sfixed(4 downto -13); --Q4.13

      signal deltaVo    :     sfixed(-6 downto -48); --Q-6.48
      signal deltaIL    :     sfixed(-9 downto -51); --Q-9.51

      --Valores auxilaries salida
      signal Ilaux      :     sfixed(3 downto -48); --Q3.48
      signal Voaux      :     sfixed(6 downto -47); --Q6.47

      signal VoauxRes   :     sfixed(6 downto -10); --Q6.10
      signal VgMux      :     sfixed(6 downto -10); --Q6.10

      --Constantes
      constant dt       :     real  := 50.0e-9;
      constant C        :     real  := 1.0e-4;
      constant dtCR     :     real  := dt/C;
      constant dtC:     sfixed(-10 downto -34):= to_sfixed(dtCR,-10,-34);
      constant L        :     real  := 9.0e-4;
      constant dtLR     :     real  := dt/L;
      constant dtL : sfixed(-14 downto -38):= to_sfixed(dtLR,-14,-38);
```

```
      --Señales tipo real
       signal IncrementoIL     :     real;
       signal IlAuxReal        :     real;

begin
       process(Reset, Clk)
       begin
             if (Reset = '1') then
                   Voaux <= (others => '0');
                   Ilaux <= (others => '0');
             elsif (rising_edge(Clk)) then
                   Ilaux       <= resize(Ilaux + deltaIL, ILaux);
                   Voaux       <= resize(Voaux + deltaVo, Voaux);
             end if;
       end process;

       --Calculo operaciones intermedias
       IDelta      <= resize(Ilaux,Ir)-Ir;
       VoDelta     <= VgMux - VoauxRes;

       deltaIL <= VoDelta*dtL;
       deltaVo <= IDelta*dtC;

       IncrementoIL      <= to_real(VoDelta)*to_real(dtL);
       IlAuxReal         <= to_real(Ilaux) + IncrementoIL;

       VgMux       <= Vg when (Q1='1' and Q2='1') else resize(-Vg,VgMux);

       VoauxRes    <=  resize(Voaux, VoauxRes);

       --Asigancion valores entrada y salida
       Il    <= resize(Ilaux, 3,-13);
       Vout  <= VoauxRes;
end Behavioral;
```

C Código configuración interrupciones

```
XScuGic IntcInstancePtr;      /*   Instanciación   del   controlador   de
interrupciones */

/*****************************************************************************
**********************/
/**
*
* Esta función configura el sistema de interrupciones para que puedan
producirse tanto en
* la UART como en el timer. Es una función específica de la aplicación.
El usuario debe
* modificarla para que se ajuste a la aplicación.
*
*****************************************************************************
**********************/

int configIntC()
{
      int Status;

      XScuGic_Config *IntcConfig;

      /*
       * Inicializar el driver del controlador de interrupciones para que
este preparado
       * para usarse.
       */
      IntcConfig = XScuGic_LookupConfig(INTC_DEVICE_ID);
      if (NULL == IntcConfig) {
            return XST_FAILURE;
      }

      Status = XScuGic_CfgInitialize(&IntcInstancePtr, IntcConfig,
                                     IntcConfig->CpuBaseAddress);
      if (Status != XST_SUCCESS) {
            return XST_FAILURE;
      }


      Xil_ExceptionInit();

      /*
       * Conectar el handler del controlador de interrupciones con el de
la lógica hardware
       * de interrupciones en el procesador.
       */
      Xil_ExceptionRegisterHandler(XIL_EXCEPTION_ID_IRQ_INT,
                        (Xil_ExceptionHandler)XScuGic_InterruptHandler,
                        &IntcInstancePtr);
}
```

D Código configuración UART

```
/*
 * El mapeo de las siguientes constantes a los parametros del XPAR se
hace en el
 * fichero "xparameters.h". Se definen aquí para que el usuario pueda
cambiar
 * todos los parametros necesarios de una sola vez.
 */

/*UART*/
#define UART_DEVICE_ID		XPAR_XUARTPS_0_DEVICE_ID
#define INTC_DEVICE_ID		XPAR_SCUGIC_SINGLE_DEVICE_ID
#define UART_INT_IRQ_ID		XPAR_XUARTPS_1_INTR

/*
 * La siguiente constante controla la longitud del buffer que se va a
recibir
 * por la UART.
 */
#define TEST_BUFFER_SIZE	64

XUartPs UartPs	;		/* Instanciación del Dispositivo UART. */

/* El siguiente buffer se usa para recibir datos por la UART. */
static u8 RecvBuffer[TEST_BUFFER_SIZE];

/*****************************************************************
**********************/
/**
*
* Esta función es la encargada de realizar el procesamiento de datos
desde la UART. Un
* contexto de interrupción se encarga de llamarla. La cantidad de
procesamiento debe ser
* mínima.
*
* Esta función proporciona un ejemplo de como controlar datos para el
dispositivo y
* es específica de la aplicación.
*
* @param	CallBackRef contiene una referencia de devolución de llamada
desde el controlador,
*		en este caso es el puntero de instanciación para el driver
XUartPs.
* @param	Event contiene el tipo específico de los datos.
* @param	EventData contiene el número de bytes enviados o recibidos
por los datos enviados
*		y recibidos.
*
* @return	Nada.
*
* @note		Ninguna.
*
*****************************************************************
*********************/

void Handler(void *CallBackRef, u32 Event, unsigned int EventData)
{
	/* Todos los datos se han recibido. */
```

```
	if (Event == XUARTPS_EVENT_RECV_DATA) {
		TotalReceivedCount += EventData;
	}

	/*
	 * Se han recibido datos, pero no el número esperado de bytes, un
	 *  timeout solo indica que los datos se detuvieron durante 8
caracteres.
	 */
	if (Event == XUARTPS_EVENT_RECV_TOUT) {
		TotalReceivedCount += EventData;
	}

	/*
	 * Se han recibido datos con un error, mantener los datos pero
determinar
	 * que tipo de errores se han producido.
	 */
	if (Event == XUARTPS_EVENT_RECV_ERROR) {
		TotalReceivedCount = EventData;
		TotalErrorCount++;
	}

	/*
	 * Se han recibido datos con un error de paridad, de ruptura o de
traza, mantener los datos
	 * pero determinar que tipo de errores se han producido. Específico
de Zynq Ultrascale+
	 * MP.
	 */
	if (Event == XUARTPS_EVENT_PARE_FRAME_BRKE) {
		TotalReceivedCount = EventData;
		TotalErrorCount++;
	}

	/*
	 * Se han recibido datos con un error de desbordamiento, mantener
los datos
	 * pero determinar que tipo de errores se han producido. Específico
de Zynq Ultrascale+MP.
	 */
	if (Event == XUARTPS_EVENT_RECV_ORERR) {
		TotalReceivedCount = EventData;
		TotalErrorCount++;
	}
}

//GLOBALES UART
XScuGic InterruptController; /*   Instanciación   del   Controlador   de
Interrupciones. */
XScuGic *IntcInstPtr = &InterruptController;

XUartPs *UartInstPtr = &UartPs;
u16 DeviceId = UART_DEVICE_ID;
u16 UartIntrId = UART_INT_IRQ_ID;

int Status;
XUartPs_Config *Config;
int Index;
u32 IntrMask;
int BadByteCount = 0;
```

```
int configUART()
{
	/*
	 * Se inicializa el driver de la UART porque ya está listo para
usarse.
	 * Buscar en la tabla config la configuración, después inicializar.
	 */
	Config = XUartPs_LookupConfig(DeviceId);
	if (NULL == Config) {
		return XST_FAILURE;
	}

	Status = XUartPs_CfgInitialize(UartInstPtr, Config, Config-
>BaseAddress);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

	XUartPs_SetBaudRate(UartInstPtr, 115200);

	/* Comprobar la construcción del hardware */
	Status = XUartPs_SelfTest(UartInstPtr);
	if (Status != XST_SUCCESS) {
		return XST_FAILURE;
	}

	/*
	 * Conectar la UART al subsistema de interrupciones de manera que
puedan
	 * producirse interrupciones. Esta función es espécifica de la
aplicación.
	 */

	//FUNCION INTERRUPCION DE LA UART
	Status = XScuGic_Connect(&IntcInstancePtr, UartIntrId,
				(Xil_ExceptionHandler)
XUartPs_InterruptHandler,
				(void *) UartInstPtr);
		if (Status != XST_SUCCESS) {
			return XST_FAILURE;
		}

	/* Habilitar la interrupción para el dispositivo. */
	XScuGic_Enable(&IntcInstancePtr, UartIntrId);

	//AQUI TERMINA FUNCOIN INTERRUPCION DE LA UART

	/*
	 * Configurar los controladores de la UART que serán llamados desde
el contexto
	 * de interrupción cuando los datos se hayan recibido, especificar
un puntero a la
	 * instanciación del driver de la UART como la referencia de
devolución de llamada
	 * para que los controladores puedan acceder a los datos de la
instanica.
	 */
	XUartPs_SetHandler(UartInstPtr, (XUartPs_Handler)Handler,
UartInstPtr);
```

```
	/*
	 * Permitir la interrupción de la UART para que puedan ocurrir las
interrupciones,
	 * configurar un bucle invertido para que los datos enviados sean
recibidos.
	 */
	IntrMask =
		XUARTPS_IXR_TOUT | XUARTPS_IXR_PARITY | XUARTPS_IXR_FRAMING |
		XUARTPS_IXR_OVER | XUARTPS_IXR_TXEMPTY | XUARTPS_IXR_RXFULL |
		XUARTPS_IXR_RXOVR;

	if (UartInstPtr->Platform == XPLAT_ZYNQ_ULTRA_MP) {
		IntrMask |= XUARTPS_IXR_RBRK;
	}

	XUartPs_SetInterruptMask(UartInstPtr, IntrMask);

	XUartPs_SetOperMode(UartInstPtr, XUARTPS_OPER_MODE_NORMAL);

	/*
	 * Establecer el timeout del receptor. Si no está establecido, y
los últimos bytes
	 * de datos no activan la interrupción entera o de over-water, los
bytes no van a
	 * recibirse. Está desactivada por defecto.
	 *
	 * En la configuración de 8 el tiempo de espera terminará despues
de 8 x 4 = 32 tiempos
	 * de caracteres. Incrementar el tiempo de espera si la tasa de
baud es alta, disminuir si
	 * es baja.
	 */
	XUartPs_SetRecvTimeout(UartInstPtr, 8);
}
```

E Código configuración Timer

```
/*
 * El mapeo de las siguientes constantes a los parametros del XPAR se
hace en el
 * fichero "xparameters.h". Se definen aquí para que el usuario pueda
cambiar
 * todos los parametros necesarios de una sola vez.
 */

/*TIMER*/
#define TIMER_DEVICE_ID		XPAR_XSCUTIMER_0_DEVICE_ID
#define INTC_DEVICE_ID		XPAR_SCUGIC_SINGLE_DEVICE_ID
#define TIMER_IRPT_INTR		XPAR_SCUTIMER_INTR

/* Tiempo entre interrupciones en milisegundos */
#define TIMER_LOAD_VALUE_ms	1

/*****************************************************************
**********************/
/**
*
* Esta función es la encargada de realizar el procesamiento de datos
desde el Timer. Un
* contexto de interrupción se encarga de llamarla al terminar el contador
del Timer.
*
* @param	CallBackRef es un puntero a la funcion "callback", devolución
de llamada.
*
* @return	Ninguno.
*
* @note		Ninguna.
*
*****************************************************************
**********************/

static void TimerIntrHandler(void *CallBackRef)
{
	XScuTimer *TimerPtr = (XScuTimer *) CallBackRef;

	/*
	 *	Comprueba si ha terminado el contador del timer; la
coomprobación no es necesaria ya
	 * que es la razón por la que esta función se ejecuta, esto solo
muestra como la referencia
	 * de devolución de llamada puede usarse como un puntero a la
instancia del contador que ha
	 * terminado, incrementar una variable compartida para que el hilo
de la ejecución
	 * principal pueda ver el timer terminado.
	 */
	if (XScuTimer_IsExpired(TimerPtr)) {
		XScuTimer_ClearInterruptStatus(TimerPtr);
		TimerExpired++;
		timeout=1;
	}
}

//GLOBALES TIMER
```

```
XScuTimer_Config *ConfigPtr;
#define  TIMER_DEVICE_ID  XPAR_XSCUTIMER_0_DEVICE_ID
XScuTimer TimerInstancePtr;
u16 TimerIntrId = TIMER_IRPT_INTR;
int LastTimerExpired = 0;

int configTimer()
{
            /*
             * Inicializar driver el Scu Timer Privado.
             */
            ConfigPtr = XScuTimer_LookupConfig(TIMER_DEVICE_ID);

            /*
             * Aquí es dónde se usaría la dirección virtual, pero este
ejemplo utiliza una dirección
             * física.
             */
            Status      =      XScuTimer_CfgInitialize(&TimerInstancePtr,
ConfigPtr,
                                    ConfigPtr->BaseAddr);
            if (Status != XST_SUCCESS) {
                  return XST_FAILURE;
            }

            /*
             *  Realización  de  una  autoprueba  para  comprobar  que  el
hardware se ha construido correctamente.
             */
            Status = XScuTimer_SelfTest(&TimerInstancePtr);
            if (Status != XST_SUCCESS) {
                  return XST_FAILURE;
            }

            /*
             * Conectar el dispositvo al subsitema de interrupciones para
que las interrupciones puedan
             * aparecer.
             */

            Status = XScuGic_Connect(&IntcInstancePtr, TimerIntrId,

      (Xil_ExceptionHandler)TimerIntrHandler,
                                    (void *)&TimerInstancePtr);
            if (Status != XST_SUCCESS) {
                  return Status;
            }

            /*
             * Habilitar las interrupciones en el dispositivo.
             */
            XScuGic_Enable(&IntcInstancePtr, TimerIntrId);

            /*
             * Habilitar las interrupciones para el modo timer.
             */
            XScuTimer_EnableInterrupt(&TimerInstancePtr);

            /*
             * Habilitar las interrupciones en el Procesador.
             */
```

```
            XScuTimer_DisableAutoReload(&TimerInstancePtr);
            /*
             * Cargar el registro del contador del timer.
             */
            XScuTimer_LoadTimer(&TimerInstancePtr,
getContador(TIMER_LOAD_VALUE_ms));

            XScuTimer_Stop(&TimerInstancePtr);
}
```

F Código actuación Timer para evitar desincronizaciones

```
while(1)
      {
            int bytesRecibidos;

            /*
             * Empezar a recibir datos antes de enviarlos porque hay
loopback, ignorando el
             * número de bytes recibidos y el valor de retorno ya que
sabemos que será cero
             */
            bytesRecibidos = TotalReceivedCount;
            XUartPs_Recv(UartInstPtr, RecvBuffer, 0);
            XUartPs_Recv(UartInstPtr, RecvBuffer, 6);

            int flagTimerStart;

            flagTimerStart = 0;
            timeout = 0;

            while (1) {

                  if(TotalReceivedCount >= bytesRecibidos+6)
                  {
                        XScuTimer_Stop(&TimerInstancePtr);
                        flagTimerStart = 0;
                        break;
                  }
                  else if(TotalReceivedCount > bytesRecibidos)
                  {
                        if (flagTimerStart == 0)
                        {
                              //Start timer
                              XScuTimer_LoadTimer(&TimerInstancePtr,
getContador(TIMER_LOAD_VALUE_ms));
                              XScuTimer_Start(&TimerInstancePtr);
                              flagTimerStart = 1;
                        }

                        //  Si  flagTimerStart  =  1  y  además  si  timer
expirado, aborto operación
                        if((flagTimerStart == 1) && (timeout==1))
                        {
                                    timeout = 1;
                                    xil_printf("Timeout. \r\n");   // solo
en modo pruebas
                                    break;
                        }
                  }

            }
```